University of Calgary

PRISM Repository **https://prism.ucalgary.ca**

The Vault Open Theses and Dissertations

2021-07-21

Detection of Emergent Behaviour in Distributed Software Systems using Data Analysis Techniques

Slama, Anja

Slama, A. (2021). Detection of Emergent Behaviour in Distributed Software Systems using Data Analysis Techniques (Master's thesis, University of Calgary, Calgary, Canada). Retrieved from https://prism.ucalgary.ca.
http://hdl.handle.net/1880/113679
Downloaded from PRISM Repository, University of Calgary

UNIVERSITY OF CALGARY

Detection of Emergent Behaviour in Distributed Software Systems using Data Analysis Techniques

by

Anja Slama

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN ELECTRICAL ENGINEERING

CALGARY, ALBERTA

JULY, 2021

© Anja Slama 2021

Abstract

Distributed Software Systems (DSS) and their sub-category, Multi-Agent Systems (MAS), are composed of several collaborative components working towards a common goal. Requirement engineering involves the consideration of competing needs and concerns for the proper presentation of the specification of software systems. The analysis of the Scenario-Based Specifications (SBS) has many important advantages to minimize the generation of unexpected behaviours in DSS. These behaviours are known as Emergent Behaviours (EB), and can potentially lead to irreversible, costly damages. In this thesis, we focus on analyzing the software behaviours from the SBS to detect EBs. The verification of the software behaviours in the early stages of software development can detect and prevent unwanted behaviours. In this thesis, different methodologies that aim to detect EBs are discussed. This thesis aims to provide a new automated, homogenous methodology to detect EBs based on their common cause of occurrence. Subsequently, different algorithms were proposed to detect the types of EBs, and examples were presented to explain the algorithms. With the adoption of data analysis techniques, we contribute to preventing these behaviours and ensuring system quality. To evaluate the proposed methodology based on the analysis of the SBS, we used two different approaches: (1) the conventional way by comparing the proposed methodology against related works' methodologies, (2) the dynamic analysis of system traces, which requires the simulation of the SBS to aggregate the system behaviour in runtime. Results show a higher efficiency of the proposed methodology in detecting EBs compared to similar work, which was also proven by the statistical modelling of the case studies' traces. Additionally, we verified the efficiency of the proposed methodology using sequential pattern mining techniques. This thesis contributes to the research of requirement engineering specifically and software engineering generally by providing an automated methodology to analyze the SBS as a black box. It assists the reusability of components and design sustainability. Moreover, the early identification of the cause of EBs empowers the designer and the software development team to handle these EBs and aids in decreasing the system cost.

Preface

This thesis is submitted for the degree of Master of Science at the University of Calgary. This thesis is an original work by Anja Slama under the supervision of Dr. Behrouz Far, except where references are made to previous work. This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

Part of this work has been presented in the following publications:

1. **A. Slama**, F. H. Fard, and B. Far, "A Hybrid System for Detection of Implied Scenarios in Distributed Software Systems (S),"Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering, vol. 2018, no. June, pp. 360–396, 2018.

2. **A. Slama**, Z. S. H. Abad, and B. Far, "Detection of Asynchronous Concatenation Emergent Behaviour in Multi-Agent Systems," Agents and Multi-Agent Systems: Technologies and Applications 2021 Smart Innovation, Systems and Technologies, pp. 77–88, 2021.

Acknowledgements

I would especially like to thank my supervisor Dr. Behrouz H. Far, who encouraged and directed me. He has taught me more than I could ever give him credit for here. I am extremely grateful for his help and support. I would also like to thank my committee members for their time.

To my parents Saida & Kamel, for their love and support throughout my life.

To my husband Imed, who has been a constant source of support and encouragement.

To my daughters Fatima & Sophia, whom I am truly grateful for having in my life.

Table of Contents

Abstract ii

Preface iii

Acknowledgements iv

Dedication v

Table of Contents vi

List of Figures and Illustrations viii

List of Tables ix

List of Symbols, Abbreviations and Nomenclature x

1 Introduction 1

1.1 Background and Motivation 1
1.2 Problems and Challenges 3
1.3 Objectives 4
1.4 Methodology 5
1.5 Research Contributions 7
1.6 Evaluation 7
1.7 Thesis Structure 7

2 Literature Review 9

2.1 Introduction 9
2.2 Scenario-Based Specifications 9
2.3 Approaches for the Detection of Emergent Behaviour in DSS 10
2.4 Sequential Pattern Mining 15
2.5 Dynamic Analysis and DSS 16
2.6 Definitions 19
2.7 Summary 20

3 Methodology 21
3.1 Introduction 21
3.2 Rational and Problem Formulation 21
3.3 Different types of EBs 24
3.4 Model-based Analysis 26
3.4.1 Pre-processing 26
3.4.2 Analysis Phase 28
3.4.3 Reporting Phase 39
3.5 Dynamic Analysis 39
3.5.1 Statistical Modelling 40
3.5.2 Pattern Mining Analysis 41
3.6 Conclusion 42

4 Case studies and Results 43
4.1 Introduction 43
4.2 Tool Support 43
4.3 Case Study Analysis 44
4.3.1 Case Study 1: Real-Time Fleet Management System 44
4.3.2 Case Study 2: Boiler Control System 48
4.3.3 Comparative Study 52
4.4 Dynamic Verification 54
4.4.1 Statistical Modelling 56
4.4.2 Sequential Pattern-Mining Analysis 58
4.5 Threats to Validity 61
4.6 Conclusion 62

5 Conclusion and Future Work 64

Bibliography 67

A Permission of copyright material 81

List of Figures and Illustrations

3.1 Phases of the proposed methodology 24

4.1 GUI of the automated tool 45
4.2 Diagram output of the automated tool 46
4.3 MSC1: Calculates bus arrival times and notify users as illustrated in [1, Fig. 2] 47
4.4 MSC2: Recalculate bus arrival times according to traffic updates as illustrated in [1, Fig. 3] 47
4.5 MSC3: Recalculate bus arrival according to manually entered data as illustrated in [1, Fig. 4] 48
4.6 The results of the analysis of the real-time fleet management system using our tool 49
4.7 hMSC and MSCs presenting the boiler control system as illustrated in [2, Fig. 1] 50
4.8 The result of the analysis of the boiler control system using our tool considering synchronous communication 50
4.9 The result of the analysis of the boiler control system using our tool considering asynchronous communication 51
4.10 Number of EBs/ISs detected in our work versus related work for different case studies 54
4.11 Correlation between the number of EBs detected and the SBS 55
4.12 Case study 1: Results of comparing hypothesis using Hyptrails 57
4.13 Network graphs presentation of H1 in (a), H2 in (b), H3 in (c) and H4 in (d) 58
4.14 Case study 2: Results of comparing hypothesis using Hyptrails 59
4.15 RC ISs detected in real-time fleet management system 60
4.16 RC ISs detected in real-time fleet management system 60
4.17 The RC ISs detected in real-time fleet management system 60
4.18 SS ISs detected in real-time fleet management system 61
4.19 Network graph presenting the behaviour of the Boiler System using sequential pattern mining algorithm 62

List of Tables

3.1 Types of EBs 25

4.1 The number of scenarios and components in the different case studies 52
4.2 The number of EBs/ISs detected in each case study 53

List of Symbols, Abbreviations and Nomenclature

Symbol or abbreviation	Definition
AC	Asynchronous Concatenation
CB	Common Behaviours
DSS	Distributed Software System
EB	Emergent Behaviour
FSM	Finite State Machine
FSP	Finite State Process
gRPC	gRPC Remote Procedure Calls
hMSC	high-level Message Sequence Chart
IS	Implied Scenarios
JADE	Java Agent DEvelopment Framework
LTL	Linear Temporal Logic
MSC	Message Sequence Chart
MAS	Multi-Agent System
NLBC	Non-Local Branching Choice
RC	Race Condition
RDC	Respond to Different Components
REST	REpresentational State Transfer
SBS	Scenario-Based Specifications
SPMF	Sequential Pattern Mining Framework
SS	Shared States
SUP	SUPport
UID	Unique IDentifier

Chapter 1

Introduction

1.1 Background and Motivation

Due to the continued improvement in performance, resource sharing and availability, the demand for DSS is increasing in a wide variety of domains such as intelligent transportation, industrial automation, biomedical instrumentation, data resources integration and Internet of Things (IoT) [3] [4] [5] [6].

Scenario-based specifications are widely used to model the messages exchange between the different components of DSS. However, due to the lack of central control of DSS and the partial nature of scenario-based specification, DSS may exhibit new behaviours at runtime that are unspecified behaviours in the scenario-based specification. These behaviours are known as Emergent Behaviours (EB) or Implied Scenarios (IS) [7] [8]. These behaviours can potentially lead to unknown system states and costly damages [9].

The early detection and correction of faults may significantly affect the cost of large software products. An undetected fault in the requirements and design may spread through the software's development phases. Therefore, identifying and improving any issue in the early design stage is more cost-effective than fixing it later and allows the prediction of later verification activities [10].

Consequently, several techniques focus on the analysis of software behaviour at various steps of software development. The three approaches to manage EBs, other than the conventional testing, are (a) the model-based analysis, which analyses the design documents [11] [12]; (b) the static analysis, which analyses the software code [13] [14]; and (c) runtime analysis, which analyzes execution traces of the system [15] [16].

There are several methods to relate requirements to system design, code and behaviour, such as Requirement Traceability Matrix (RTM) that relates requirement to the requirements artefacts and Unified Modeling Language (UML)'s class diagrams that relate them to concrete design [17] [18]. In this research, we consider the relation between the requirements and system behaviour. A practical approach to describe the complicated behaviour of DSS, given its requirements, is the use of SBS. Message Sequence Charts (MSCs) and UML's interaction diagrams are generally used to specify scenarios of interactive systems [19].

Scenarios model the interaction between system components. The scenarios will be the source to verify the conformity of the system behaviours to the system requirements.

Consequently, this research will investigate EBs in DSS and MAS based on the analysis of the system model. Among the common problems in this area, we mention the following:

(P1) Each type of EB is detected using a different approach.

(P2) The process of detection of EBs is not fully automated.

(P3) Investigating efficiency of the model-based analysis at run time is generally omitted in previous research.

Taking into consideration these problems, we have implemented a new methodology for the detection of EBs. This methodology detects EBs by investigating the conditions that lead to each type of EB's occurrence in the run time. The discovery of the cause of the EBs is more efficient than detecting EBs and then investigating their cause as the same weakness in the design may lead to several ISs. Moreover, we investigated the implemented methodology's efficiency by analyzing a use case's trace.

In this research, we aim to provide answers to the following questions:

(Q1) Are we able to propose a fully automated methodology that allows the detection of common EBs?

(Q2) How efficient is the proposed methodology?

The main phases of our research are:

- Phase 1: Identify the different types of EBs in DSS

- Phase 2: Propose a new methodology for detecting different types of EBs from scenario-based specification and proposing general solutions for each type of EB based on previous research.

- Phase 3: Automate the detection of EBs by implementing the proposed methodology.

- Phase 4: Investigating the efficiency of our methodology.

1.2 Problems and Challenges

The tendency to use DSSs in the computing industry leads to the study of its behaviours and related issues. The main reasons which make the design and implementation of DSSs an arduous task are their openness, the number of parallel processing, and unprofessional implementation [20].

In DSS, the system components are distributed across space, while they communicate via a communication network. The interaction between these components may result in undesirable EBs during the system execution, which in general, are implied by an undecidable action of the system components.

The DSS performance issues may be found during designing, testing and monitoring the software. This research focuses on detecting EBs in an early phase of the software development, which results in an improved quality and reduced cost. Additionally, discovering software failure at an early stage may prevent major system maintenance and avoid delaying progress of the final project.

Formal methods are verification techniques to discover EBs. These mathematical meth-

ods require knowledge and expertise to check the correctness of the system modelling with respect to the domain properties. Among problems encountered in detecting EBs during the design phase while using automaton theory for sophisticated design are that defining special algorithms or languages requires unique expertise and is time-consuming.

Model checking is used to ensure that the system behaviour fulfills its requirements. This methodology compares the model (e.g., state machines) and specification language, which show the system behaviour. The challange in model checking is the state explosion problem [21], which is due to the exponential increase of the global states of the system as the number of the processes grows [22].

In order to avoid the problem stated in previous approaches, we have adopted a message content independent method. This method requires less effort to verify the modelling even when the system requirements change because it does not require the change of the formal specification. While, model checking requires specifying the formal method to formally verify the algorithm, we check the system behaviour by the analysis of the scenario-based specification. This method differentiates between send and receive messages while analyzing the system behaviours. Furthermore, this proposed methodology detects and eliminates the cause of the EBs, which consecutively, eliminates all its subsequent ISs.

1.3 Objectives

Considering the aforementioned problems, our research aims to detect potential EBs either for planned software or while implementing the next release of the software. We assume that the update of the system's design may introduce new EBs. Our methodology tackles the lack of knowledge in the early design stage by considering the software components as black boxes. The early detection of EBs enable us to discover the weaknesses of the design and propose solutions. Additionally, a simulation is used to certify the components' behaviours and the whole system. Our objective is to analyze the scenario-based specification for detecting and

depicting conditions that may cause EBs. Proposing an effective solution based on the cause of EBs may imply preventing the appearance of a group of ISs caused by a common source.

In this research, we will address the following research questions:

Q1: What are the shortcomings of previous methodologies?

Q2: How to define an efficient and fully automatic methodology to detect different types of EBs?

Q3: How to use the system tracing analysis of DSS and MAS to prove the existence of EBs?

Our primary research goals are:

Goal 1: Introduce a new homogeneous methodology for the detection of common types of EBs based on their cause of occurrence.

Goal 2: Implement the proposed methodology to automate the detection process and assist the designer and the developer in later phases of the software life cycle.

Goal 3: Certify the system behavior through its dynamic analysis.

1.4 Methodology

The early detection of potential EBs is required to support the development team to avoid software failure in a more effective and timely manner. Previous research studies have investigated the verification of the correctness of the behaviours of DSSs through identification and classification of EBs [23]. Classification of EBs, based on their cause of occurrence, results in preventing EBs to occur during the runtime. This reference classification catalogue helps to identify the cause of EBs and provides a general solution for each type.

Studying the conditions in SBS that cause EBs allows their direct detection from the scenarios without converting to intermediate models. Therefore, to identify EBs, we opted not to convert SBS into transition models (e.g. interaction graphs or state transitions). As this methodology does not require regeneration of graphs for the analysis, which may require

specific languages and domain expertise, it supports the multi release of the software, which is necessary to meet the increasing market requirements.

The proposed independent message methodology aims to detect specific patterns in the interaction diagrams that may exhibit unexpected behaviours during the run-time. Additionally, it aims to identify EBs defined in previous research, which are revisited in our research. The required information is extracted from the scenarios for both component and system-level analysis.

In enourmous systems, traditional methodologies which use state transition may face shortage or computation problems while analyzing the system behaviours. Thus, we considered the analysis of SBS using pattern mining techniques. This technique allows for the detection of sequential patterns, even for extensive data. Consequently, the EB detection models are independent of the software specification size.

We are implementing our methodology into a fully automated tool to detect different types of EBs proposed in the catalogue of EBs. The methodology does not require expert intervention to identify the source of the EBs. Our goal is to support developers and testers to identify potential EBs and avoid possible failures as it is impossible to test all possibilities in sophisticated software. The methodology allows for the detection of potential EB, which can be positive or negative, according to the requirements specification documents. Both positive and negative EBs are unexpected, however, positive EBs do not contrast with the software requirements, whereas negative EBs can negatively impact the system's service and lead to failure. This distinction requires expert domain knowledge. Thus, the classification of EBs as negative or positive is left to the stakeholder and the designer.

The spreading of the fault at execution time may depend on different phases that the system passes through before the testing. Thus, we opted for the simulation of the software systems to verify the proposed methodology's accuracy.

1.5 Research Contributions

The main contributions in this thesis are:

1. A new methodology for the analysis of SBS.

2. The implementation of our methodology to automate the detection of different types of EBs.

3. The analysis of the simulated software's traces to verify the correctness and efficiency of the proposed methodology.

1.6 Evaluation

The evaluation of the detection of EBs is performed by analyzing the results obtained by applying the proposed methodology on case studies extracted from related works which are considered as a reference in this field. Since there are no benchmarks or datasets to evaluate this methodology other than checking the proposed methodology against other works, we opted to adopt a dynamic analysis approach to verify the efficiency of the proposed methodology. The dynamic analysis approach is used to analyze the trace collected by the simulation of the SBS of the case studies. The result of these evaluations is presented in Chapter 4.

1.7 Thesis Structure

The rest of this thesis is organized as follows. Chapter 2 introduces different approaches for the detection of EBs and an overview of the related research studies. Chapter 3 introduces different types of EBs and presents a new methodology to detect them. Chapter 4 presents the application of the proposed methodology to various case studies. Finally, the last chapter

presents our conclusions, limitations we faced in this work and our recommendations for future studies.

Chapter 2

Literature Review

2.1 Introduction

EBs are unwanted behaviours that show during the system's execution and do not conform to the system requirements. There have been different approaches for the verification and validation of software behaviours. In this chapter, we will review various approaches for the detection of EBs based on the SBS. Furthermore, we will present several research areas related to this work, such as sequential pattern mining, dynamic analysis and DSS frameworks.

2.2 Scenario-Based Specifications

Requirement engineering allows an accurate understanding of the system's needs and stakeholder's perspectives and develops them into detailed requirement [24]. In addition, the documentation of systems' behaviours is essential in all companies for system improvement and maintenance [25]. Moreover, the modelling of the elicited requirements enables the verification and validation of the system. For instance, the availability of the models early in the requirement process allows model execution and code generation [26]. Thus, careful behavioural modelling is crucial for DSS.

Although using state machine models to describe the dynamic system behaviour presents several benefits, these models present a labour-intensive task [27]. An alternative technique to synthesis the system behaviours is the use the SBS. SBS such as Message Sequence Chart (MSC) and UML sequence diagrams are commonly used to represent the requirements specifications of systems [28]. An MSC is a graphical language standardized at the ITU [29]. The equivalent of MSCs in the UML are sequence diagrams [26]. Both behavioural models present the scenarios as a sequence of message interactions (i.e. events) among system components (i.e. agents, processes or actors). Concomitantly, the high-level message sequence chart (hMSC) shows the possible paths between the MSCs composing the system. It allows the ordering of the scenarios for proper system functionality. The equivalent of hMSC in the UML is the Interaction Overview Diagram.

An SBS models the ordered interactions between independents system components and the system's reaction with external events [28]. It states the intended behaviours of the system to provide its functionalities. However, some behaviours of the system may not be visually seen in these diagrams. In DSS, SBS may present a partial description of the software behaviour due to its system's components' concurrent behaviours [8]. Thus, we aim to provide an approach to synthesis the analysis of the SBS and to support in the verification of software systems by the early detection of potential EBs.

2.3 Approaches for the Detection of Emergent Behaviour in DSS

An EB is an unexpected behaviour that shows in the runtime, whereas it does not conform to software requirement specifications [30] [31]. System verification and validation are processes used to ensure system quality. They are crucial to avoid failures that may result in costly and considerable damages while meeting system users' needs. [32]. Verification is used to check if the system meets its specification. It is intended to verify if the system requirements

and specifications have been met. It is usually conducted using modelling, simulation and testing throughout the system's lifecycle [33]. The validation is used to ensure that the system meets the stakeholders' needs. It aims to affirm that algorithms are performing their intended functions effectively.

In this thesis our focus is on distributed system verification. The fundamental approaches to system verification are model-based analysis, static analysis and runtime analysis. Examples from each approach are presented in the following:

- Model-based analysis: This approach considers the analysis of model-based specifications. Besides presenting a convenient form of documentation, models facilitate the verification of the software systems at an early stage [34, 35, 36, 37].

- Static analysis: This approach analyses the code and does not require program execution. It explores all possible program behaviours and can find bugs that do not necessarily impact runtime behaviour. It is specific to the programming language used during the system implementation [38, 39, 40, 41].

- Runtime analysis: This approach checks behavioural constraints against system execution traces and logs. It is limited to the observed behaviour occurring during the runtime of the existing application [42, 43].

The three mentioned approaches are complimentary for the verification of the software. Besides the importance of runtime analysis and static analysis to avoid software failures after implementing the system, the early detection and removal of faults in the requirement and design phase may significantly reduce the project cost [44]. Our research is in the scope of the model-based analysis.

The model-based analysis allows to check if the critical system properties such as safety properties, time constraints, deadlock freedom, or the models' proper implementation are satisfied. Common research on model-based analysis uses formal methods for system verification [9, 45, 46, 47]. Formal methods use mathematics and formal logic to prove the model's

correctness with respect to a given specification that the system should satisfy [48].

Among the model-based verification algorithm relying on formal methods, we mention model-checking. Model-checking involves building a finite representative model of the system and verifying if the expected system behaviour satisfies the given specification [33]. Thus, model-checking comprises the model's construction, defining the formal specification corresponding to the desired system behaviour and then the verification [49]. If the system does not meet the specification, a counterexample is generated. The study of the counterexample allows pinpointing the source of the error in the model. Among the research based on model checking, we cite [50, 51, 52]. Model-checking can be automated, but it may lead to a space explosion during the system behaviours' analysis [22]. Besides, the verification procedure may take time while analyzing the models because of the large number of states. One of the main limitations to the adoption of model checking is that the modelling language is generally specific to the model-checking tool. Moreover, this method focuses on the detection rather than the analysis of the cause of failure, which requires further analysis to identify the cause of the problem [28].

The verification of software design to detect the faults that may lead to runtime EBs in DSS is required. In this research, we target assuring DSS's quality by verifying the system's behaviour anticipated from the models. Our analytical approach is not based on formal verification. Model-based verification is a verification technique that assures the proper implementation of the system. The verification of the system's models allows detecting the weakness that may lead to runtime EBs in DSS to prevent their occurrence. However, manual verification of the system models and their review is not always possible, depending on the DSS's complexity and scale. In DSS, the geographic distribution and the growth of the system in complexity may contribute to the occurrence of EBs during the system execution [53]. In the following, we present the approaches presented in related work to detect EBs.

Uchitel et al. [54] presented a technique to generate the architecture models using Labeled Transition Systems (LTSs). In addition, they proposed an algorithm to generate safety

properties. In order to detect ISs, the synthesized model is checked against the safety properties. In [2], Uchitel et al. proposed the definition of specification rules for verifying system behaviours. These rules are defined using Finite State Process (FSP) language, which is a formal language. The definition of these constraints requires domain expertise. This proposed approach provides a counterexample in case an issue is detected in the requirements specification. Among the limitations of this approach are the consideration of only synchronous communication between the system component and the state explosion problem.

In our previous study [55], we used the LTSA-MSC to model the software behaviours in the form of MSCs. This tool allows the transformation to Finite State Process. Finite State Process is a notation that generates a Labelled Transition System (LTS). This latter is a Finite State Machine (FSM) used to represent the exchanged messages for each component. It is a directed graph presenting the transition between the states for each component. The LTS of the system is obtained by the parallel composition of the system components. This parallel composition presents the trace of the system that we considered for the analysis of software behaviours. The work's result was a hybrid system that allows the detection of component level implied scenarios based on traces obtained by the LTSA-MSC. The identification considers the catalogue of EBs provided by Fard [23]. However, we considered the change in the research path to avoid the state explosion problem encountered during the analysis and to consider asynchronous communication.

De Melo et al. [56] propose a methodology that consists of analyzing the implied scenario produced by the LTSA-MSC tool proposed by Uchitel. The methodology consists of collecting and grouping the implied scenarios into Common Behaviours (CBs) which is a group of ISs having similar messages. Then, the clustering of similar CBs into a family of common behaviours allows the classification of behaviours as negative or positive. However, it is important to mention that the definition of constraints in these processes are manual. These last steps of the methodology require domain knowledge. The definition of FSP constraints is the proposed solution to solve EBs. This methodology is limited to the analysis

of synchronous communication.

To specify the system's behaviours, Alur et al. [57] methodology was based on MSCs and Message Sequence Graph (MSG). The model is linearized and checked against an automation showing the unwanted behaviour for each component to check the system's behaviour. The authors investigated the realizability of the system and classified it as strong and weak. However, there are some issues related to this methodology. For example, this methodology cannot be implemented fully automatically as it requires expert knowledge to model the system.

Moshirpour's research is based on semantic causality. The semantic casualty means that receiving an event should happen after the corresponding sending event and that for an event to happen, it requires the accomplishment of all casual procedures first [7]. This methodology was automated in a software tool. However, the use of FSM for behaviour modelling may lead to state space explosion during the system analysis.

Other research considered the verification of the system's behavioural for the detection of EBs by investigating the cause and the condition in the modelling that may lead to the occurrence of EBs instead of the model checking techniques.

For example, Muccini's approach aims to identify the ISs cause by the non-local choices [58]. The proposed algorithm allows identifying the process that generates an IS in the non-local choice node but it is not implemented into a tool. However, this approach assumes only the synchronous communication between the system components.

Song et al. [59] [60] converted the SBS into the specification and implementation graphs. The difference between the two graphs presents the unenforceable orders. They show that the existence of unenforceable orders may cause ISs. This approach is used to detect the cause of the ISs under the asynchronous and synchronous communication styles. The difference between the specification and the implementation graphs requires investigation in order to prevent ISs.

Fard [23] considered each scenario's presentation in a separate graph to preserve the

interaction information for each component and between components. Each graph presents the process interactions in the MSCs or sequence diagrams. The graph used to analyze the system's behavioural is created based on two concepts; the Core and the Node. The Core contains information for the component level, whereas each Node contains the state of the component and its interactions with other components. The proposed methodology is message content-independent. The main contribution of Fard's work was the modelling and the classification of EBs. However, general algorithms were presented only for component-level analysis.

To verify complex and concurrent interactions in DSS, we considered the analysis of the components' behaviours based on the cause and conditions that may lead to the occurrence of EBs. In this work, we study the common types of EBs presented in previous research. Therefore, we propose a new categorization of the EBs presented by Fard [23] for a better analysis of the system behaviour. This catalogue is the result of the study from the literature of common issues in DSS and MAS. It also provides solution repositories for the detected problems.

2.4 Sequential Pattern Mining

Specification mining considers the use of a mining algorithm to detect frequent interaction patterns from execution traces. It has been applied to solve problems in different domains such as healthcare, text mining, web usage mining, etc. [61] [62] [63] [64].

Among the kinds of frequent patterns mining, we name sequential patterns mining. Sequential pattern mining is used to retrieve relevant patterns from sequential data while considering the order of items [65].

Different sequential pattern mining algorithms were used to analyze sequential data such as GSP [66], PrefixSpan [67], Spam [68], etc. GSP uses an Apriori-like approach to discover sequential patterns. GSP applies a level-wise search to discover patterns. Among the limi-

tation of GSP is the repeated scan of the database to calculate a candidate's support, the possibility of generation of non-existent pattern and the massive amount of memory used for the generation of patterns [69]. PrefixSpan is a pattern growth method focusing on searching the prefix sub-sequence and projects only their postfix sub-sequences on the database. However, the repeated scan of the database and the projection into a projection database can be costly in memory and runtime [70]. Spam is an Apriori–all type of algorithm. It uses pruning techniques to generate candidate sequences and improve the performance of the algorithm. It is more efficient than PrefixSpan in a larger dataset. CM-Spade [71], uses vertical representation to generate the patterns and calculate their support. It applies pruning techniques to overcome the performance bottleneck of vertical algorithms. To enhance the performance of CM-SPADE method, MCM-SPADE was introduced to be used on a multi-core processor system [72].

Each sequential pattern mining algorithm has characteristic features to generate candidate sequences. For example, all super-sequences of a closed pattern should have smaller support, whereas all the maximal pattern's super-sequences should not be frequent. In this research, we aim to discover closed contiguous sequential patterns. The contiguous constraint applies strictly to maintaining the original adjacency of items. CCSpan [73] adopts a snippet-growth paradigm for the generation of candidates and three pruning techniques. This algorithm is efficient and scalable in terms of database size and support threshold. Besides considering the contiguous constraint, it precisely records the pattern's occurrences in the database while avoiding the presentation of redundant patterns.

2.5 Dynamic Analysis and DSS

Undesirable behaviours are not explicitly presented in the specification of the system. However, it may occur in the implementation of this later. These EBs are generally the result of implementing the action from the local level of the component due to DSS's characteri-

zation. The EBs can appear after several runs of the system. For this reason and to prove our approach's correctness, we opted for the dynamic analysis of the traces.

Dynamic analysis is the process of testing software at runtime in a virtual or real environment. It allows the identification of issues that may arise in runtime. Among dynamic software analysis approaches, we name runtime verification. Runtime verification allows the checking of the correctness of the system behaviours at runtime. It considers the observation of the execution traces of the system and checks it against property specifications. This latter is extracted from the specifications in the form of formal language. The comparison of property specification against the execution traces ensures the system's behaviours conform to its desirable behaviour [74]. In general, the verification of the execution sequences is processed by state machines [75].

To validate our methodology using the system's dynamic analysis, we ensure the ISs detection according to our methodology occurs at the execution time. The generation of the traces of the DSS under analysis is a crucial phase. The accuracy of the monitoring affects the accuracy of the analysis.

DSS is composed of different components located in different physical locations and communicate through messages to achieve a common goal. The adoption of distributed systems architecture allows a higher availability, scalability and performance improvements of the software. However, to ensure the integration of the data and the system's expected functionality, other management and conditions must be considered to ensure the systems' proper functionality. Among the different types of distributed system architectures, we cite client-server, three-tier, n-tier and peer to peer. Different platforms have been proposed to facilitate the implementation of DSS. Among these platforms that allow easy implementation of distributed software systems, we name REST (REpresentational State Transfer).

REST is an architectural style. It allows an independent implementation of the server. The server and the client communicate through the use of resources instead of commands. However, REST communication is synchronous in nature.

Another modern platform has gained enormous tendency called gRPC (gRPC Remote Procedure Calls). It is a RPC framework that allows defining a transparent communication between the client and the server. Among the differences between gRPC and REST is the payload, which makes it faster than REST. Moreover, gRPC calls can be synchronous or asynchronous.

MAS are DSS composed of agents that cooperate to achieve a common goal [76]. JADE (Java Agent DEvelopment Framework) is an open-source framework fully implemented in Java that allows building MAS. It uses asynchronous message passing based on FIPA (Foundation for Intelligent Physical Agents) standards between agents.

In this thesis, we opted for JADE to simulate the execution of our system for several reasons. JADE facilitates the implementation of interacting entities to perform the functionalities of the system. Furthermore, entities can behave as servers by providing services to other entities or clients by invoking other entities' services. Besides, the exchanged interactions between the system entities are conforming to the FIPA software standards specification.

To reason about the dynamic scenarios, Statistical Model Checking (SMC) proposes the analysis of the trace of a specific number of runs of a simulated system. It aims to prove the conformity of the system behaviour to the specification using a statistics technique [77].

Traces of the software system present massive data that require detailed analysis. Thus, the use of statistical models to extract analytical information.

Among the approach to statistical modelling, we name Bayesian Statistics [78]. It applies the Bayes theorem, which uses conditional and normal probabilities. In this thesis, we used HypTrails [79], a Bayesian approach used to compare hypotheses, to analyze the system traces.

Hyptrails is a general approach that uses the Markov Chain [80] to model human trials and Bayesian inference to compare a set of hypotheses, presenting the belief about the transition between the states, while considering empirical observations [79]. The evidence presented the probability of the data given a hypothesis.

2.6 Definitions

In this section, we present the definition of the keywords used in work:

Definition 2.1. Message Sequence Chart (MSC):

It describes the interaction among parallel components of a distributed system. It was introduced in International Telecommunications Union (ITU–T) Recommendation Z.120.

Definition 2.2. High-level Message Sequence Charts (hMSC):

It is a set of MSCs. It combines MSCs and represents their execution order. It allows getting the high-level structure of the process.

Definition 2.3. Scenario:

It is an interaction diagram. It could be sequence diagrams or MSCs.

Definition 2.4. Visual Order:

It is the local order of all processes and set of all edges.

Definition 2.5. Process:

It is active if the first action of the process is of type send in the local order of a scenario specification.

Definition 2.6. Ready process:

In a synchronous system, it is a process that has no interaction within the scenario. However, a ready process in an asynchronous system may only receive interactions.

Definition 2.7. Shared states:

They are similar states of a component in different scenarios.

Definition 2.8. Vectors ϑ:

ϑ is a set of interactions. A vector v is an interaction presented by the association of the Sender component UID, the interaction label and the Receiver component UID.

Definition 2.9. Receiver vector:

It is defined as the association of the interaction label and the Receiver component ID.

Definition 2.10. Feasible path:

It is a traceable path according to the system hMSC from a start scenario to an end scenario.

In this research, we may refer to the DSS component as a component, process or agent. Moreover, we refer to unexpected behaviours by Emergent Behaviours or Implied Scenarios.

2.7 Summary

The purpose of this chapter was to present the related literature. In the next chapter, we will introduce a new methodology for the detection of common EBs in SBS.

Chapter 3

Methodology

3.1 Introduction

In this chapter, we aim to highlight common issues in the distributed system design. A novel methodology to detect EBs is presented, which includes algorithms used to identify particular patterns in the SBS that may lead to potential EBs. These unforeseen issues have to be fixed by the designer or taken into consideration during the system's implementation to prevent unintended behaviour at the execution time.

3.2 Rational and Problem Formulation

EBs can occur at the component level or system level. The component-level EB is generally the result of concurrent communication between system components and the loss of information about the interacting components. The system-level EB may be resulted from the combination of different components behaviours which may result in a new implied scenario.

The detection of EBs requires analyzing each component's behaviour separately and also in interaction with other components. Thus, we considered the interaction diagram's analysis using a novel methodology to detect different types of EBs. The detection methodology focuses on detecting EBs related to the ordering of events rather than the causal ordering

and is based on locating the causes that contribute to the generation of EBs during the runtime.

The main contribution of this work is to use the mining algorithm to detect patterns in the scenario specification. Contrary to the majority of traditional behavioural modelling, our methodology does not require transformation of SBS to another modelling form, such as the state model, to detect EBs. The proposed methodology considers the analysis of the information used to model the system behaviour for detecting different types of EBs. Accordingly, the proposed methodology can emphasize on EBs in the interaction graph provided by the designer.

In this work, we analyzed the interaction process based on the data extracted from the visual representation of the system interaction (i.e. MSCs or sequence diagrams). Thus, the same modelling is considered for component and system-level analysis.

The interaction diagrams are converted to events, which present the components' interactions while preserving the local and global interactions order. Each event is a combination of the sender, the message label, and the receiver of the message. The events are then analyzed for a check of patterns and cases that may cause EBs.

The catalogue of EBs considered in this work defines the causes of EBs. Our goal is to discover the sets, from the provided SBS, that match the patterns which may lead to an unwanted system's behaviour. The necessary information used for the analysis of the system behaviours is a triple t=(s, m, r) as a single word, where s denotes the unique identifier of the sender component, m denotes the message label or the method invoked for the communication between s, and r denotes the unique identifier of the message type, order and scenario holder, as well as the receiver of the message. The information used for the detection of each type of EB may be different and extra information may be used for the detection of specific types.

This methodology was implemented into a homogeneous and fully automated tool to consistently identify the cause of EBs and annotate them in the scenarios.

The proposed methodology can be performed in two main phases, as shown in Figure 3.1:

- Model-based Analysis: This involves analyzing the system model-based specifications to detect unconformable behaviours to the system requirements. This main phase of this work consists of three tasks:
  - Pre-processing Phase: The data are derived from the design specifications and converted to a list of events to be used in the next phase.
  - Analysis Phase: In this phase, the data is analyzed to detect typical patterns used to identify EB's different types.
  - Reporting Phase: The analysis results are exported to stakeholders in textual and visual format to support them in decision-making.
- Dynamic Analysis: This phase aims for verification of our proposed methodology during the runtime. In this phase, two techniques were adopted (i.e. dynamic analysis and statistical modelling). These techniques highlight the components' interactions during the system execution and identify ISs.

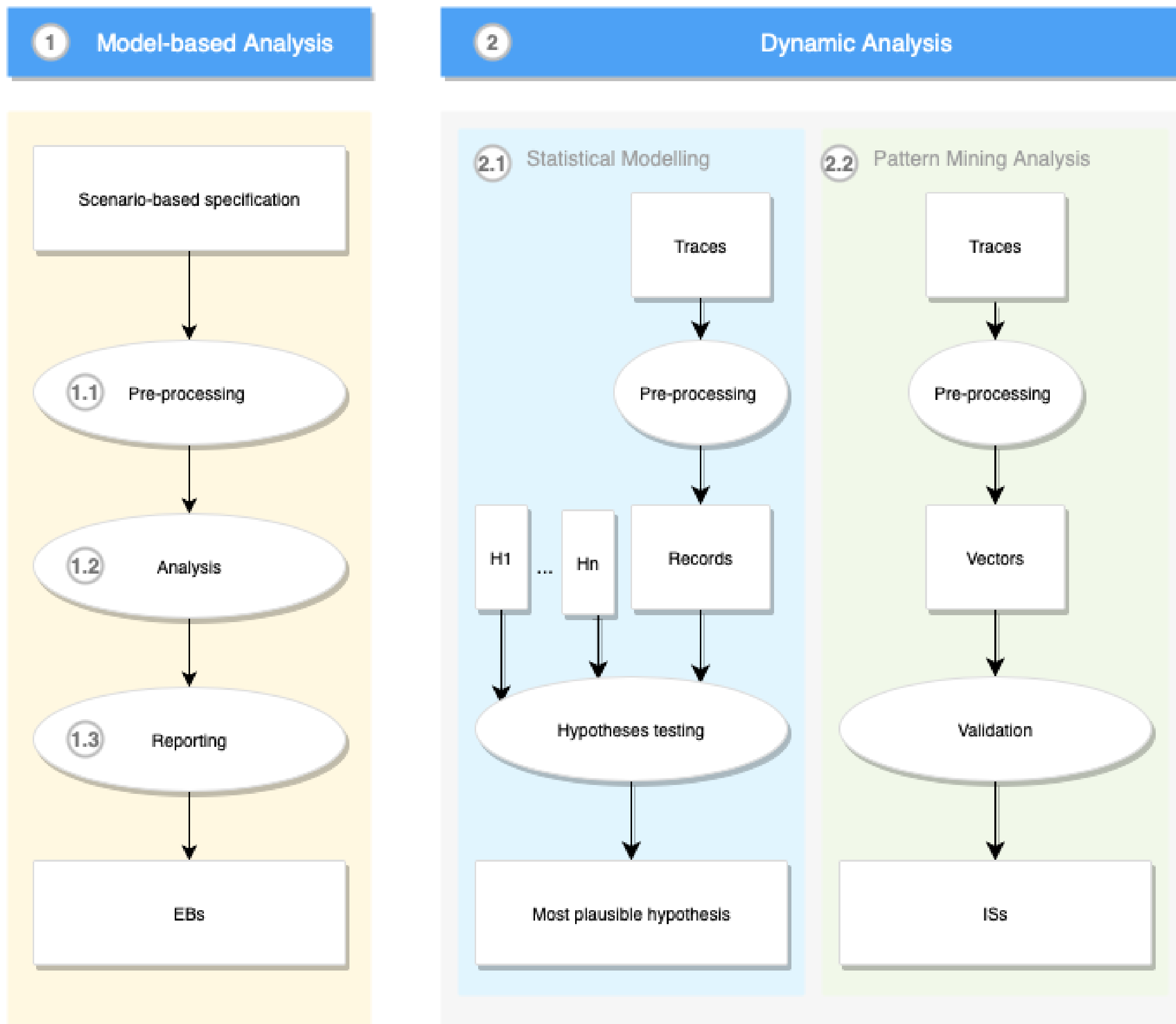

Figure 3.1: Phases of the proposed methodology

3.3 Different types of EBs

Different types of EBs have been presented in previous studies [60][2][81]. The analysis of the system modelling for the detection of EBs was either at the component level or system level.

In this research, we opted for categorizing EBs based on the common cause of the EBs. This categorization defines more general subcategories that can make the resolution of the EBs easier for the designer. This classification of EBs may lead to a decrease in the algo-

rithm's complexity during the detection of EBs and ease their prevention. The introduced catalogue of EBs is presented in Table 3.1.

Type of EBs	**Component Level**	**System Level**
Identical States	• Shared States • Respond to Different Components	
Unenforceable orders	• Race Conditions	• Non-Local Branching Choices • Asynchronous Concatenation

Table 3.1: Types of EBs

The following presents a brief description of the EBs studied in this research. Details about each type of EB will be given with greater focus in the next sections.

1. Identical States: This category of EB considers the analysis of unwanted behaviours related to shared states. We refer to shared states as a sequence of identical states in different system scenarios.

    (a) Share States (SS): This EB is resulted from the non-deterministic behaviour of the component after its identical states between different scenarios. The system may show an EB if the subsequent state after the identical state is sending a message, and there are no defined restrictions.

    (b) Respond to different components (RDC): This EB presents a particular case of shared states EB where the sender of the message is not considered during the scenario analysis for detecting shared states. The component may not have access to message sender information for privacy reasons or in case of the loss of the data.

2. Unenforceable orders: This category of EBs is caused when the interactions between the system components are uncertain and not having a restriction on the active processes

in different scenarios.

(a) Race conditions (RC): refers to a situation when a change of the order of received messages may alter the system's behaviour. In other words, a permutation in the order of occurrence of the events may alter the expected program output.

(b) Non-Local Branching Choice ($NLBC$): Unexpected behaviour of the system may occur when the component can choose between different branches in the higher-level interaction diagram. A non-local branching choice is identified when the first events at the branching points are sent by different processes.

(c) Asynchronous Concatenation (AC): Based on the high-level Message Sequence Chart (hMSC), AC implies the assembly of the scenarios along a lifetime, which may disclose unexpected behaviours.

3.4 Model-based Analysis

3.4.1 Pre-processing

To reduce the amount of processed information and to decrease the computational complexity of the methodology and its corresponding implementation, we opt to adopt two pre-processing phases, as described below.

Pre-processing for the Detection of Neutral Components

This phase helps to reduce the number of components to analyze and, consequently, reduces computational complexity. A neutral component is a component that does not contribute to the emergence of unwanted behaviours. These components are eliminated in this phase.

Based on the catalogue of EBs presented in Table 3.1, the properties that contribute to the emergence of unwanted behaviours are sent messages by the component under analysis for identical states EBs or active interactions for the unenforceable order EBs. Therefore,

we may consider those neutral components that do not contribute to the appearance of unexpected behaviours as passive components that only receive messages and not send any.

Algorithm 1 is used for the detection of neutral components. The definition of vectors ϑ can be found in section 2.8. For the detection of neutral components, we extract vectors for each component of the system where this latter is sending messages to other components using the function *getSendingInteractions (interactionDiagrams, C)* (lines 1-2). If the extracted vector list is empty, then the component is appended to the list of neutral components N (line 4).The components identified as neutral are considered safe and are omitted from future analysis.

Algorithm 1: Neutral components detection algorithm

Input: Interaction diagrams
Output: Neutral components N

```
1 for each component C
2     ϑc_s ← getSendingInteractions(interactionDiagrams, C )
3     if ϑc_s = ∅
4         append(N, C)
5     end if
6 end for
7 return (N)
```

Pre-processing for Pattern Mining

In this pre-processing phase, we transformed the scenario-based specification raw data into an understandable format by the sequential patterns algorithms used in later phases to detect closed contiguous sequential patterns.

The first step was the extraction of relevant information to our analysis into a sequence database. Thus, we opted to discard some information for less expensive computation. Nevertheless, we considered the creation of new features for future use in pattern mining. For example, we created a new feature composed of the triplet {Sender UID, Message label, Receiver UID}. Sender UID and Receiver UID are unique identifiers, each of which refers to all components of the same type within the system.

The sequence database format is defined as each line is a sequence of items presenting the interactions within a scenario, and each item is a word stating an interaction presented by the combination {Sender UID, Message label, Receiver UID}.

In this work, we used pattern mining techniques, such as CCSpan [73], to identify interesting and useful patterns rather than a manual analysis of the data, which may be difficult and time-consuming. Consequently, we converted the textual data to numerical data according to the SPMF format, where each word is substituted with a unique ID. The file used as the input of a data mining algorithm is a list of sequences where -2 presents the end of the sequence. Each item is presented by a positive number and each sequence is a list of items separated by -1.

The pre-processing of our textual data allows for a faster and less expensive pattern recognition task. In this case, frequent patterns are interactions that co-occur in different scenarios. Although, further analysis using algorithms presented in the next section 3.4.2 should be applied to detected EBs.

3.4.2 Analysis Phase

This work analyzes the system's interaction diagrams to detect common types of EBs in DSS and MAS. MAS is a sub-class of DSS. In this work, we detect components and system-level EBs. However, we classify the EBs according to their level and the leading cause of occurrence for two reasons. The first reason is that the same modelling can be used to detect EBs for both components or system levels. The second reason is that the analysis, according to the EB's primary cause of the occurrence, makes the analysis less complicated. Previous research considered the study of interaction diagrams to detect EBs mainly according to their level of occurrence because each level requires different modelling for the detection of EBs.

The different types of EBs considered in this research are presented in section 3.3. Table 3.1 showed the identification of the different types of EBs according to their detection level

and their main cause of occurrence. There are component-level and system-level EBs. The component-level EB is implied by a single component's behaviour, whereas the system-level EB is implied by the behaviour of the system's different components. These types of EBs present common issues identified in the DSS by previous research. Each type of EB occurs by the combination of some reasons and conditions. More details about each type of EBs will be presented in the next sections.

For the detection of patterns that may cause unexpected behaviours, we consider using pattern mining techniques. The details of the pattern mining techniques and the algorithms used to classify the detected patterns are explained in more detail for each EB type. The input for the pattern recognition algorithm is a sequence of sentences. Each sentence represents an interaction diagram, and a sentence is a sequence of atomic blocks. As mentioned in the previous section, a pre-processing phase was required to convert the data to be fed to the data-mining library used in this research.

In this work, we considered the detection of EBs occurring at the component and system levels. We refer the type of EB to the kind of interaction or pattern on the modelling that may lead to unwanted behaviours. We considered two main branches for the detection of EBs: (1) check EBs that occur as a result of the interaction of different scenarios, (2) detect EBs that active components can cause. In the following sections, we will present the different algorithms used to detect each type of EB.

Identical States

The algorithm used to detect Identical States is defined in Algorithm 2. To detect Identical States, we first reduce the complexity of the algorithm's calculation by detecting neutral components. The latter are excluded from the system's components list to obtain the NN list considered for detecting Identical States (line 1). For each non-neutral component, we extract its interaction vectors (line 3). If the interaction vectors are part of more than one scenario (line 4), the extracted list is further considered to detect shared interactions. In order

to check the patterns (line 6), we first pre-process the input (line 5). This pre-processing phase presented in the section 7 mainly converts the interaction vectors belonging to the same interaction diagram into a sequence of sentences. The detected patterns (line 7) are then checked to detect Identical States' EBs.

Algorithm 2: Identical states detection algorithm

```
   Input: interactionDiagrams
   Output: Identical States
1  NN ← getNonNeutralComponents(interactionDiagrams)
2  for each component C in NN do
3      ϑc ← getInteractions(interactionDiagrams, C )
4      if length(getScenarios(ϑc)) > 1 then
5          sequences ←preprocessing(interactionDiagrams)
6          patterns ← checkPattern(sequences)
           // Checking shared interactions EBs
7          potentialEBs ← checkEmergentBehaviours(C, ϑc, patterns)
8      end if
9  end for
```

Shared States (SS) This EB occurs when the agent presents identical states in behavioural modelling and gets confused about what action to take [34].

Identical States are the result of merging the behaviour of the component from different scenarios. However, the state after the shared states creates branches in the behavioural model of the sender component making its behaviour non-deterministic. The component sending the message after the shared state may force the message to other components since it has the freedom of sending the message. Consequently, the component may continue its behaviour after the shared states in another interaction diagram.

At this stage, the atomic block structure comprises the sender, the message label, and the receiver, while a UID identifies each component in the system. We refer to this atomic block structure by a vector. It is necessary for information processing, and it is used to extract concealed patterns. To detect Identical States, we consider synchronous communication between the system's components. Thus, we considered the detection of closed contiguous

sequential patterns. The detection of Shared States was presented in [23] as a future trend.

To prevent this unwanted behaviour, a condition should be defined for sending the message. For instance, the component may send the message when it reaches the state to send it, whereas this state is reached by performing its tasks in another MSC.

The algorithm used to detect SS considering synchronous communication is presented in Algorithm 3. The shared interactions for each component detected using pattern mining techniques are provided as an input for this algorithm. The provided shared states as input for the algorithm are the results of applying the pattern mining technique on sequential data considering the sender, the receiver, and the label of the message. The detected patterns with support greater than one (SUP>1) present the set of shared states of the component among the system's various scenarios. In this work, SUP refers to the absolute support of a pattern in the sequential database. In other words, the support of a frequent sequence presents the number of sequences (i.e., scenario) enclosing the sequential pattern (i.e., identical states). To consider a sequence of interactions as a shared state between different scenarios, the sequence of interactions should appear in more than a scenario. Thus, the detection of identical states implies the selection of only the patterns that have support no less than one. The change in the order of the received messages is not considered during this analysis because we are considering only synchronous communication for the detection of SS. Consequently, we opted to apply a contiguous sequential pattern technique for the detection of shared states.

To detect SS, we first locate the detected patterns for each pattern in the interaction vectors of the component under analysis using the function *locateIdenticalStates (patterns, ϑc)*(lines 1-2).This function returns the start and end global order of each pattern in the scenario-based specification data. Furthermore, The support of the detected pattern presents the number of occurrences of the identical states in the different interaction diagrams. For each identical state, we verify if the sender of the next interaction after the identical state is the component under analysis (lines 3-6). The function *getNextInteraction(identicalState)* returns the next interaction of the identical states within the same scenario. If the sender of

the next interaction after the identical state is the component under analysis, the interaction is reported as an SS as the component may not determine what action to take next.

Algorithm 3: Shared states EBs detection algorithm

```
  Input: C, ϑc, patterns
  Output: SS
1 for each pattern in patterns
2     identicalStates ← locateIdenticalStates (patterns, ϑc)
3     for each identicalState in identicalStates
4         nextInteractions ← getNextInteraction(identicalState)
5         if getSender(nextInteractions) = C
6             report(SS)
7         end if
8     end for
9 end for
```

Respond to Different Components (RDC) From the local view of the component, some states are identical in more than one scenario. However, the sender of the same message could be different components. In some cases, like anonymity, the component may not consider the agent causing its state transition. In other cases, the component loses information about interacting components. Thus, the agent may be confused with which scenario to proceed correctly if the component has a sending message after the last state of the identical states [82]. Consequently, the component may oblige the sending of the message to another component.

This EB was introduced in [82]. The detection of the shared states causing the potential unwanted behaviour considers that the component under analysis loses the information about the sender and the receiver of the messages and brings it to the same state.

Considering that the component does not have a view of the interacting components, the observed patterns are based on the knowledge of the message label and its order for each component. For the detection of shared states at the component level, the sender was not considered a useful feature to be selected to detect patterns. The atomic block structure, in this case, is the message label and the receiver. To preserve the order of the interactions

between the components, we opted to use a contiguous sequential patterns technique. The retrieval of the patterns is based on the algorithm defined in [73]. In general, this algorithm allows the detection of closed contiguous sequential patterns.

Shared states present a particular response to a different component EB when the same shared interactions are also identical in terms of the sender between different scenarios. In other words, SS are included within the RDC when the message senders are the same. In this case, RDC presents a general situation of SS.

The agent's insufficient information makes it more difficult for the component to make the right decision, especially when there are several possible actions. Consequently, adding a variable that shows the sender of the message is required. Defining a restriction on the sending messages based on the defined variable prevents the agent from making the incorrect decision and continuing its processing in another MSC when receiving the same message by a different component in different MSCs.

The algorithm used to detect RDC is presented in Algorithm 4. The shared interactions for each component present the input for this algorithm. These shared interactions result from applying the pattern mining technique on sequential data. The detection of RDC requires detecting patterns in sequential data where the atomic data is composed only of the component under analysis and the message label. Only shared states with support greater than one are considered for further analysis. The support reflects the number of occurrences of the shared states in the scenarios. To preserve the order of the received messages during this analysis, we opted to apply a contiguous sequential pattern technique to detect shared states.

To detect RDC, we start by analyzing the interaction diagram data to detect recurrent patterns while assuming the component loses the information about the sender of the receiving interactions. Thus, we first locate for each pattern the detected patterns in the interaction vectors of the component under analysis (ϑc) (lines 1-2). As RDC is a general case of SS, we avoid reporting RDC already reported as SS by verifying identical states using

the method $isRDC()$ (line 3). This method consists of checking whether the interactions of the identical states are identical in different scenarios. If all interactions are identical, the RDC is a SS and $False$ is returned. Else, an RDC is reported (line 4). For each identical state, if the sender of the next interaction after the identical state is the component under analysis, then the next interaction is reported as an RDC (lines 5-8).

Algorithm 4: Respond to different components EBs detection algorithm

```
   Input: C, ϑc, patterns
   Output: RDC
1  for each pattern in patterns
2      identicalStates ← locateIdenticalStates(patterns, ϑc)
3      if identicalStates.isRDC() then
4          report(RDC)
5          for each identicalState in identicalStates
6              nextInteractions ← getNextInteraction(identicalState)
7              if getSender(nextInteractions) == C
8                  report(RDC, nextInteractions)
9              end if
10         end for
11     end if
12 end for
```

Unenforceable Order

Race Conditions (RC) In our research, we refer to a race condition when the event's occurrence in execution is different from the visual order of the events defined by the design [83]. The loss of ordering among the interactions gives rise to a difference between the system modelling and the system implementation [84]. Meanwhile, the changes in the order of the messages do not necessarily imply a race condition. The received messages' order is important to the receiver when the changes in the received messages' order alter the component's functionalities.

This type of EB occurs in two cases:

1. When a component has interactions with other components, and no condition is defined to preserve the received messages' order according to the system specification. A change

in the received events may lead to the generation of a new system's behaviour. The unexpected behaviour resulting from non-preservation of events' visual order is a race condition and has to be verified as wanted or unwanted behaviour. In this case, the analysis is restricted to finite communication behaviours, and then only active processes are considered in this analysis.

2. When a component receives the same messages (i.e., m1 and m2) in one of the interaction diagrams and receives the same messages in a different order (i.e., m2 and m1) in another interaction diagram. Even though the change in the order of the received messages respects the local order of the component in another MSC, the interacting components can differ. Consequently, a component may exhibit *RC* when it receives the same messages in the expected order but from different components than defined in the SBS.

Considering the system behaviour analysis from the system design, other factors related to this EB as message loss and delays will not be considered in this analysis. For the detection of race condition EB, we check the situation where the change in the order of messages leads to implied scenarios.

A control on the order of the received event may prevent the occurrence of this type of EB. However, this may lead to deadlocks [85]. Moreover, the existence of the possible implied scenario in the specification may prevent the occurrence of the race condition EB.

The algorithm used to detect *RC* is presented in Algorithm 5. For each scenario, we get the list of active process interactions (lines 1-3). As a concurrent communication to the same component in different scenarios may result in EB, we group the interactions by its receiver in each scenario (line 4). Then, we change the order of the received interactions for each block of interactions having the same receiver in a scenario (line 7). An *RC* (case 1) is reported if the change in the order of the receiver vectors of the permutation is included in receiver vectors of active process interactions in other scenarios (lines 13-15). Otherwise, if

the permutation of the block of interaction with the same receiver is feasible in the interaction diagram, we report an RC (case 2) (lines 9-13) .

Algorithm 5: RC **detection algorithm**

```
   Input: interactionDiagrams
   Output: RC
1  for each scenario in scenarios(interactionDiagrams)
2      blocks ← getActiveProcessesFirstInteractions(interactionDiagrams,
        scenario)
3  end for
4  candidateBlocks ← groupByScenarioReceiver(block)
5  for each candidate in candidateBlocks
6      permutations ← permutation(candidate)
7      for each potentialEB in permutations
8          case1 ← false
9          if (ReceiverVector(potentialEB) ∈ ReceiverVector(candidateBlocks))
10             if (potentialEB ∉ candidateBlocks)
11                 report(RC- case 2)
12             end if
13         else
14             case1 ← True
15         end if
16     end for
17 end for
18 if case1
19     report(RC- case 1)
20 end if
```

Non-Local Branching Choice ($NLBC$**)** This implied scenario occurs when the system's high-level structure has at least two branches, and the active process in each branch is the instance of different components. This implied scenario may appear when a process follows one branch, whereas another process follows another branch. The resulting behaviour therefore does not conform to the system specifications.

When deciding which branch to follow depends on more than a process; it is referred to as non-local branching as it is not a local decision for one component [86].

In Algorithm 6, we consider the detection of the non-local branching choice $NLBC$. First, we extract the forks of the high-level structure (line 1). Then, for each fork of the

Algorithm 6: $NLBC$ **detection algorithm**

Input: Interaction diagrams, hMSC

Output: $NLBC$

```
1  forks_h ← GetBranches(hMSC)
2  for each fork F in forks_h
3      MSCs_F ← getScenarios(F)
4      for each scenario MSC in MSCs_F
5          activeProcesses ← GetActive(MSC)
6      end for
7      if the number of unique process p_a in activeProcesses > 1
8          report(NLBC)
9      end if
10 end for
```

high-level structure, we extract the list of scenarios forming the fork and the active processes in each scenario (lines 3-5). If the number of unique active processes is greater than one, an $NLBC$ is reported (lines 7-9).

This implied scenario may be prevented by defining a global condition in the branching for all active processes.

Asynchronous Concatenation (AC) The asynchronous concatenation of scenarios occurs when components do not wait for the end of execution of other components in the current scenario and proceed with its execution in the next scenario.

This EB can be caused by active processes where no condition is specified on when to start the next scenario, and its sub-graph of the active component shows different behaviour from the system level. When a process does not interact in all the scenarios, it may present a different behaviour than the behaviour presented in the hMSC. When the process is active, it may start its actions in the next scenario according to its local behaviour without waiting for other processes.

The hMSC models the transitions between i MSCs. A feasible path is traceable from an initial scenario MSC_j to a final scenario $MSC_{\mathrm{j+1}}$ according to the hMSC, such that MSC_j precedes $MSC_{\mathrm{j+1}}$. Each scenario of MSC presents the sequence of interaction $\{m_1, m_2, ..., m_k\}$

between agents $\{a_1, a_2, ...a_l\}$. If agent a_m has interactions in MSC_m and $MSC_{\mathrm{m+n}}$ ($n > 1$) scenarios, respectively, and if it is an active agent in $MSC_{\mathrm{i+n}}$, a new path $MSC_i \rightarrow MSC_{\mathrm{i+n}}$, which is infeasible according the system hMSC, is implied.

The Algorithm 7, which we previously published in [87], is used for the detection of AC. In order to reduce the computational cost, we detect active processes in each scenario (line 1). For each active component, we detect ready scenarios (line 3). Then we detect the predecessor and the successors of the ready processes (lines 5-6). If the component is active in the successor processes and the path predecessor successor is an infeasible path, this latter path is reported as asynchronous concatenation EB (lines 7-15).

Algorithm 7: ***AC*** **detection algorithm**

Input: Interaction diagrams, hMSC
Output: Asynchronous Concatenation EB

```
1  Components_active ⟵ GetActiveComponents(interaction_diagrams)
2  for each C in Components_active
3      scenarios ⟵ GetReady(C)
4      for each scenario S in scenarios
5          successors ⟵ GetSuccessors(S)
6          predecessors ⟵ GetPredecessors(S)
7          for each successor SUC in Successors
8              if IsActive(C, SUC)
9                  for each predecessor PREC in predecessors
10                     if InfeasiblePath(PREC, SUC)
11                         report(AC)
12                     end if
13                 end for
14             end if
15         end for
16     end for
17 end for
```

In [23], they considered the detection of this type of implied scenario by comparing the component's high-level structure against the system's high-level structure. Comparing the graph does not always lead to an implied scenario. For example, the branch for an active process, it is not active in all the scenarios forming the branch. It may show a different path, but it is for passive scenarios. Moreover, if it has a part in the last interaction in the

previous MSC, it could not cause an EB as it marks the end of the MSC.

Conditions must be defined to guarantee the perseverance of the message order. For example, blocking and waiting for functions must be defined to prevent the execution of other functionalities of the component in other scenarios before other components complete their functionalities according to the scenario specification.

3.4.3 Reporting Phase

This phase consists of reporting EBs detected during the analysis of the system modelling. To obtain consistent results, analyze different system's modelling independently of their size, and ensure the system's sustainability, we opted to implement our approach into an automated software system. During the reporting phase, the automated tool presents a detailed report in both textual and graphical formats. Any potential EBs will be addressed in the system modelling or considered in later software development phases. Moreover, the obtained results are used for the dynamic analysis phase to verify the system behaviour in runtime. The implemented tool will be presented in more detail in the next chapter.

3.5 Dynamic Analysis

To examine our methodology's efficiency in detecting EB, we analyzed a case studies behaviours in runtime. The verification process was elaborated using two approaches. The first approach consisted of comparing the expected behaviours of the system according to the different hypotheses. However, the second approach considered the detection of interaction patterns between the system components.

To analyze the behaviours of a case study, we considered the system design simulation to understand its behaviour. Using this approach, we tried to answer the question of what EBs the system will show if it is implemented precisely as described in the scenario-based specification. In other words, we verified whether the behaviour described in the scenarios

is the same as the implemented DSS behaviour.

Since there are infinitely many runs of the system in general, we considered the trace of software for many runs. This step aims to prove the presence of errors indicated during the scenario-based specifications analysis. Each trace contains different information, such as the sender and the receiver of each interaction, its local order, and to which scenario it belongs.

In this task, we abstracted the essential features of the system. The event that changed the state of the system entities considering the message label. For the coding, we chose a general-purpose language (i.e., JAVA).

To verify the system behaviour in runtime, we opted to simulate one of the case studies using JADE as presented in Section 4.4. Among the reasons behind considering JADE in the simulation is the high-level of abstraction for easy implementation of MAS. For the simulation of SBS, we implemented the system's components as agents (i.e. component) that interact through message exchange. Each component has control of its actions and can decide autonomously when to perform them. Furthermore, each agent can be a server and/or a client. The component of the system presents software entities that can be located on different platforms. As in this work, we focused on the program synthesis from SBS, and we worked on the implementation of the program based on the specification without studying other factors that may impact the system behaviours like time constraint and ontology consideration.

3.5.1 Statistical Modelling

Among the types of data analysis techniques, we consider statistical modelling to gain useful information about the system behaviour using the collected traces. In this section, we use statistical modelling to visualize the relationships between the collected data and the hypotheses in a mathematical way. To determine if our tool accurately detected EBs in the system modelling, we compared the potential EBs detected using our methodology and the expected behaviour of the case study to the monitored data.

In the following analysis, we analyze the traces using the Hyptrails [79], which displays the evidence of hypotheses compared to the monitored data. Each hypothesis represents the belief about the transition between components for the component-level analysis and between scenarios for the system-level analysis. The data corresponding to each hypothesis is generated through a random walk on the Markov Model corresponding to the hypothesis. The walker starts at the initial node, presenting the first interaction of the model, and then chooses the next node by randomly selecting one of the outgoing links (i.e., next interaction) of the current node. The generated data is then compared to the tracing data to retrieve the most plausible hypothesis. The probability distribution of state transition is obtained by observing the transition from the current state to the next state.

For component level EBs as shared states, responses to a different component, and race conditions, we aim to detect the change in the order of the interactions' local order. However, for system-level EBs as asynchronous concatenations and non-local branching, we aim to detect the change in the order of the execution of the scenarios. Accordingly, we pre-processed the tracing data into a set of events to represent the transition between the scenarios.

At this level, we aim to verify the presence of EBs in the runtime.

3.5.2 Pattern Mining Analysis

As the analysis of the trace for detecting potential EBs is a data-intensive activity, we used a Closed Contiguous Sequential pattern CCSpan [73] for the elicitation of sequential patterns. In this analysis, we required a contiguous analysis of the patterns because it was important to track the order of the event's execution. For example, the race condition results from the change in the order of the received messages to a component.

In this work, we opted for the use of closed sequential pattern mining to reduce the number of the collected patterns while achieving the same expressive power. For instance, a closed pattern is a pattern that is not included in a super pattern having the same support. It avoids the presentation of inefficient and redundant patterns.

The analysis of this data is different between component and system levels. For the component level, we focus on tracking the interactions between the component and the order of the interactions' execution. For the system level, we focus on detecting the pattern of interaction between the system scenarios. Although the data structure is different in both levels, we always track the order of the execution from a Source to a Target. For this reason, we apply a sliding window of two.

3.6 Conclusion

By extracting the required information from the system modelling, we were able to detect components and system-level EBs. The adopted pre-processing phase allowed the reduction of the computation complexity. Different conditions have been considered for the detection of different types of implied scenarios. Each type of Ebs can have different conditions to be satisfied to be implied in the system or component level. A new methodology has been presented for the detection of the defined EBs.

Chapter 4

Case studies and Results

4.1 Introduction

In the previous chapters, we presented our methodology to detect common types of EBs. In this chapter, we apply our methodology to various case studies. To assess the potential of our methodology to detect EB in SBS, we used two approaches. The first traditional approach in this area is based on a comparative study. The second approach consists of the analysis of the traces resulted from the simulation of two case studies.

4.2 Tool Support

Our methodology benefits from being implemented into an automated tool, which provides faster results. The tool supports the applicability of our methodology on different case studies independent of their size.

This tool was implemented using Python, which is a powerful object-oriented tool. Furthermore, it is an open-source technology that offers access to extensive support libraries. Python showed productivity and speed besides a user-friendly structure. These criteria were essential in the implementation of our tool, which requires fast analysis of the data. For the graphical interface, we opted for PyQT, which offers a modern look and is suitable for

simple tasks.

In this work, we considered the use of the Visual Paradigm modelling tool to convert the project modelling constructs into the CSV format. Other modelling tools, such as StarUML also offer this feature, however, one of the advantages of the Visual Paradigm is that it is compatible with other modelling tools and allows importing modellings from different file types.

To automate the pattern detection, Sequential Pattern Mining Framework (SPMF) [88] was used for pattern mining. SPMF is an open-source library, which provides implementation of many pattern mining techniques. In this work, we integrate this library in our tool to detect patterns that may cause the occurrence of EBs during the execution time.

The snapshot shown in Figure 4.1 shows the graphical user interface of the automated tool. After importing the system's SBS, the user may choose which type of EB to consider during the analysis. By starting the system checking, a report outlining potential EBs is presented. The output of the analysis using our methodology is shown in the results section. The output is also displayed in the form of notes on the diagrams as displayed in Figure 4.1.

4.3 Case Study Analysis

In this section, the proposed methodology is applied to ten different case studies, formerly reported in the literature and of different complexities, to validate its efficiency in detecting EBs. Here, two case studies are presented in details:

4.3.1 Case Study 1: Real-Time Fleet Management System

The real-time fleet management system, presented in [1], allows the scheduling based on the tracking of the fleet. This system sends text messages or emails to the transit users registered to receive notification for the bus arrival. The system collects the bus location data using GPS, weather condition and traffic condition to estimate the time required to reach each the

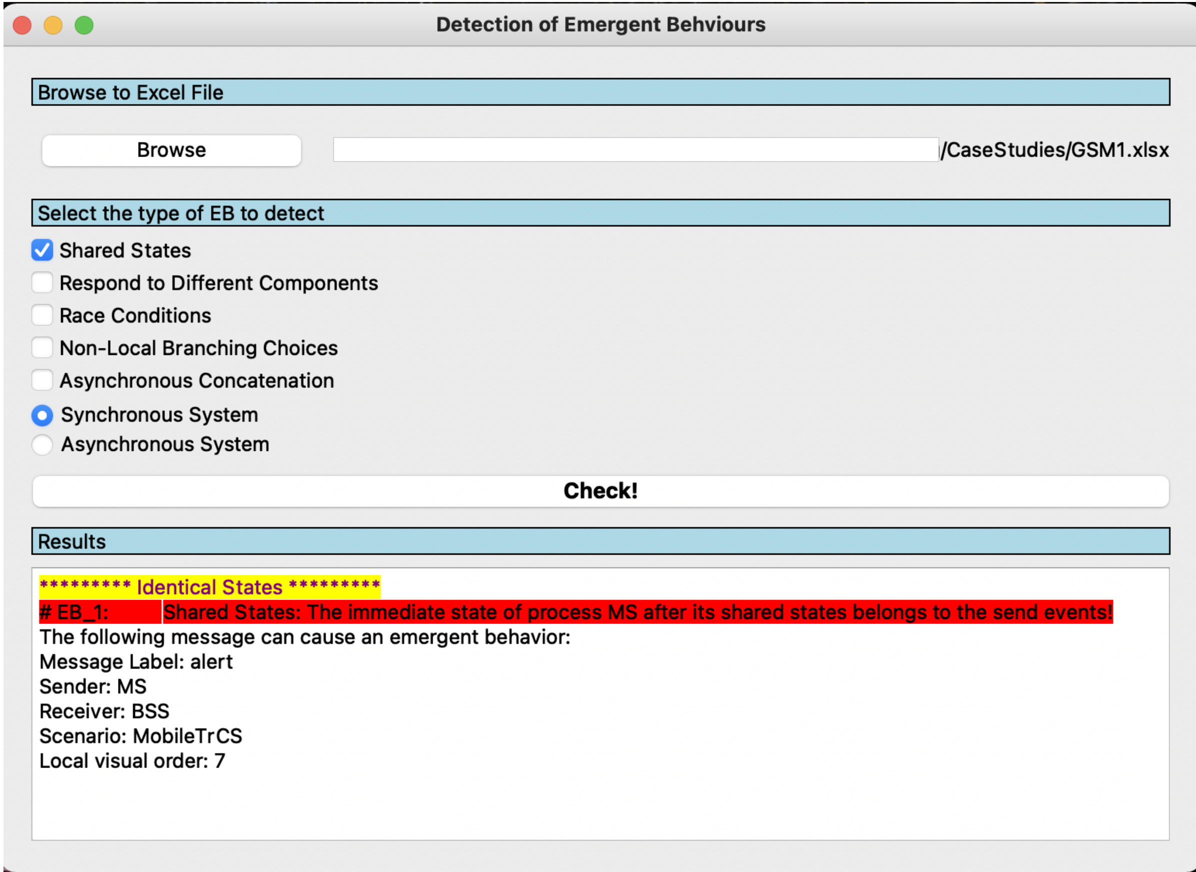

Figure 4.1: GUI of the automated tool

bus stop. The system has three scenarios: calculating bus arrival times and notifying users, recalculating bus arrival times according to traffic updates and also according to manually entered data. This has been shown in figures 4.3, 4.4 and 4.5, respectively.

The analysis of SBS of the real-time fleet management system depicts two types of EBs (i.e., *SS* and *RC*) as presented in Figure 4.6. The *update* message that the process Server sends to the *Web Interface* may cause an *SS* as it is sending a message after the identical states are detected in *MSC1* and *MSC2*. This interaction may cause an EB because the *Server* may send the *update* message after the identical states, whereas the system was supposed to wait to receive *send traffic update* and recalculate the arrival time. The system's analysis reveals more than identical states; however, not all identical states can cause EBs. For example, the identical states composed of the interactions between *estimate time*, *update*,

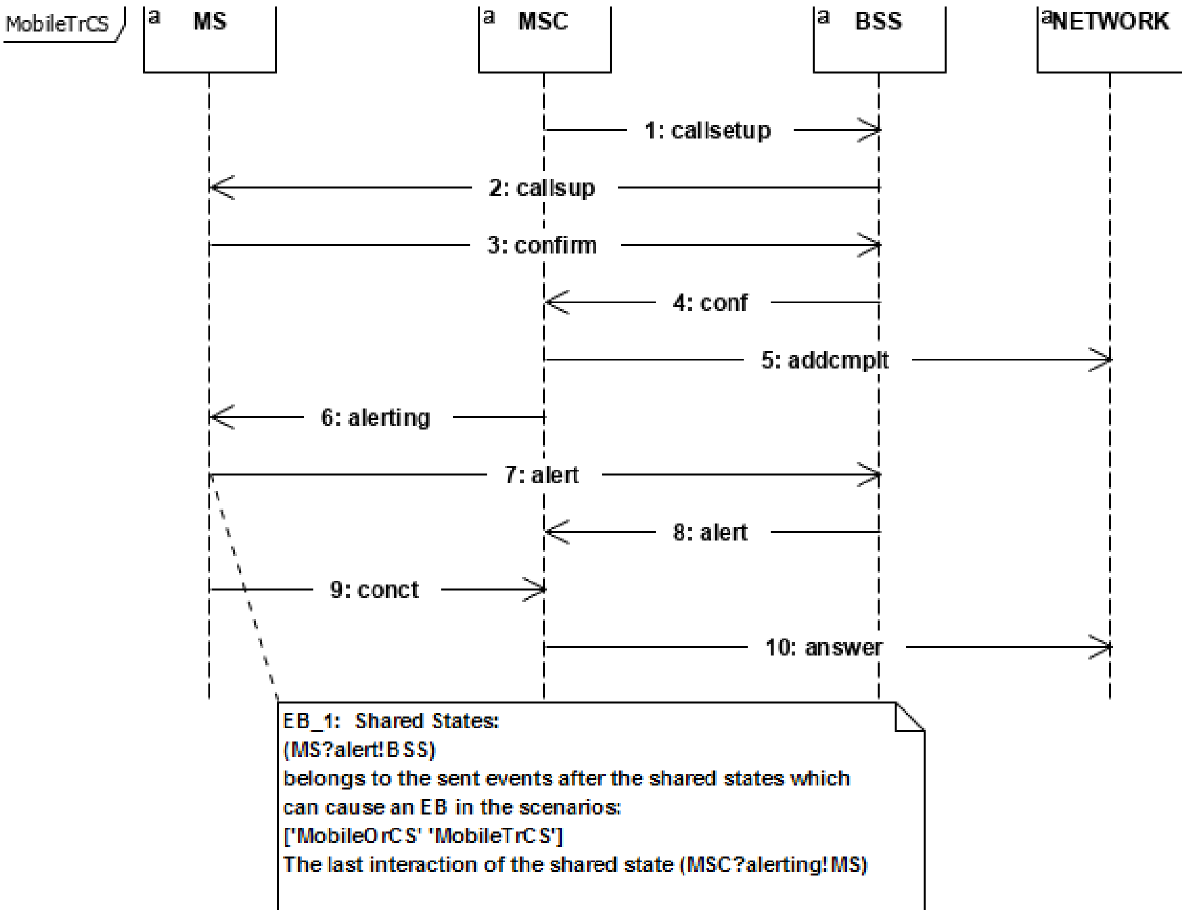

Figure 4.2: Diagram output of the automated tool

select users and *send SMS* presents the last interactions in the scenario *MSC2* and *MSC3*, and there is no interaction after the identical states, which does not lead to an EB.

The EB detection analysis, reveals three *RCs*. The first *RC* was detected in the *MSC1* and was caused by the concurrent interaction *send location data*, *send traffic data* and *send weather data* by the active processes *GPS Data Rec.*, *Traffic Data Rec.* and *Weather Data Rec.*, respectively. The same *RC* is also detected in the *MSC1*. Moreover, the change in the order of the sent interactions *send data* and *enter data* to the component *server* may cause an *RC*. This analysis reveals the possibility of the occurrence of *RC*, which is based on the domain knowledge and decision whether these interactions have to be received in the proper sequence for the system to behave as expected. In case the behaviour of the system does not depend on the specific timing of the reception of these messages, the results are

considered during the system implementation. For example, the next action of the system should not be uniquely dependent on the last received message based on the visual order of the interactions in the scenarios but the reception of all messages that may cause the *RC*.

Based on our analysis, the EBs are detected between the MSC1 and MSC2. The trace-based simulation is performed to analyze the trace of the execution of these two scenario specifications.

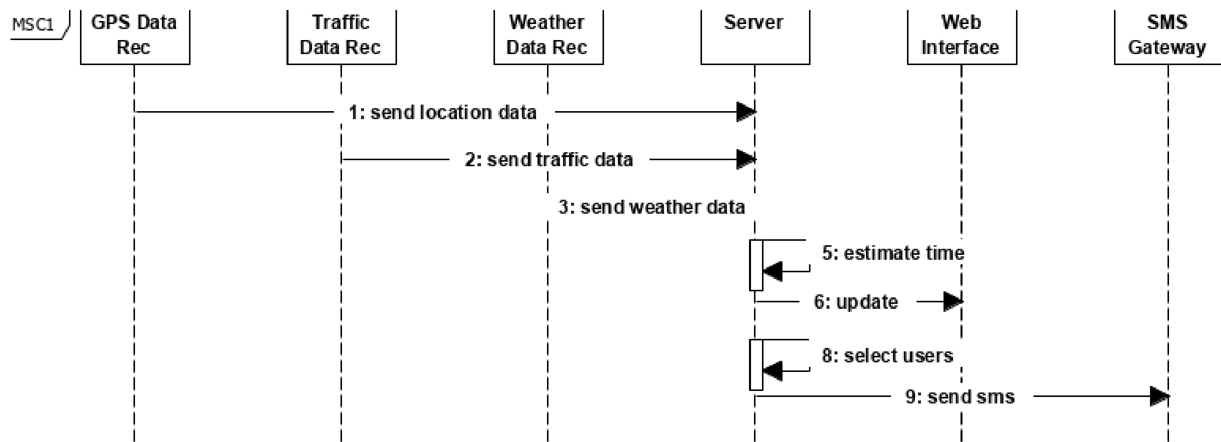

Figure 4.3: MSC1: Calculates bus arrival times and notify users as illustrated in [1, Fig. 2]

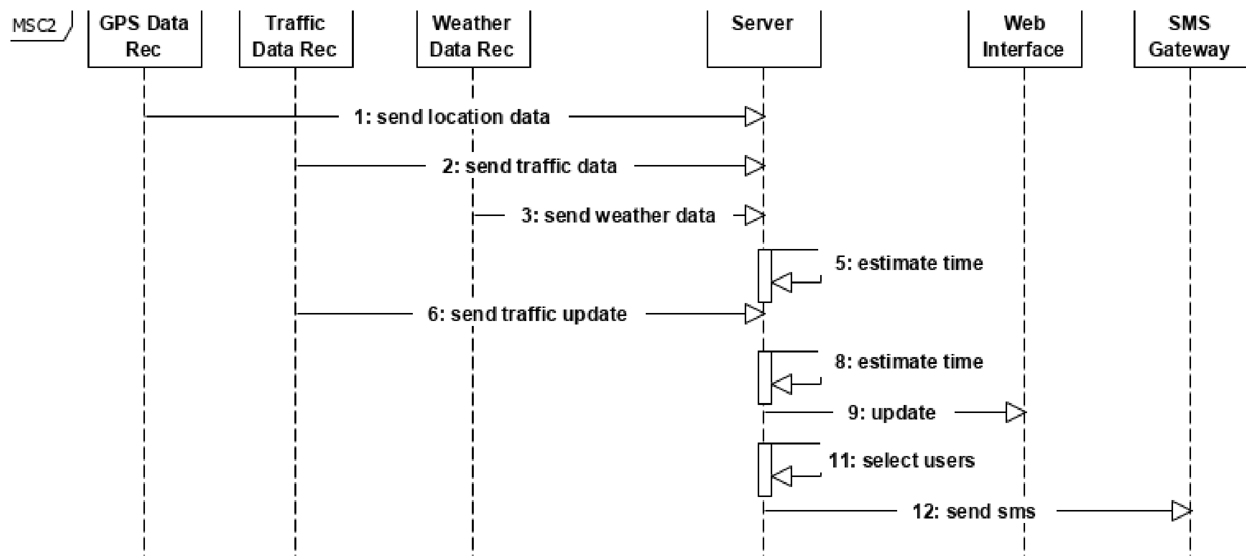

Figure 4.4: MSC2: Recalculate bus arrival times according to traffic updates as illustrated in [1, Fig. 3]

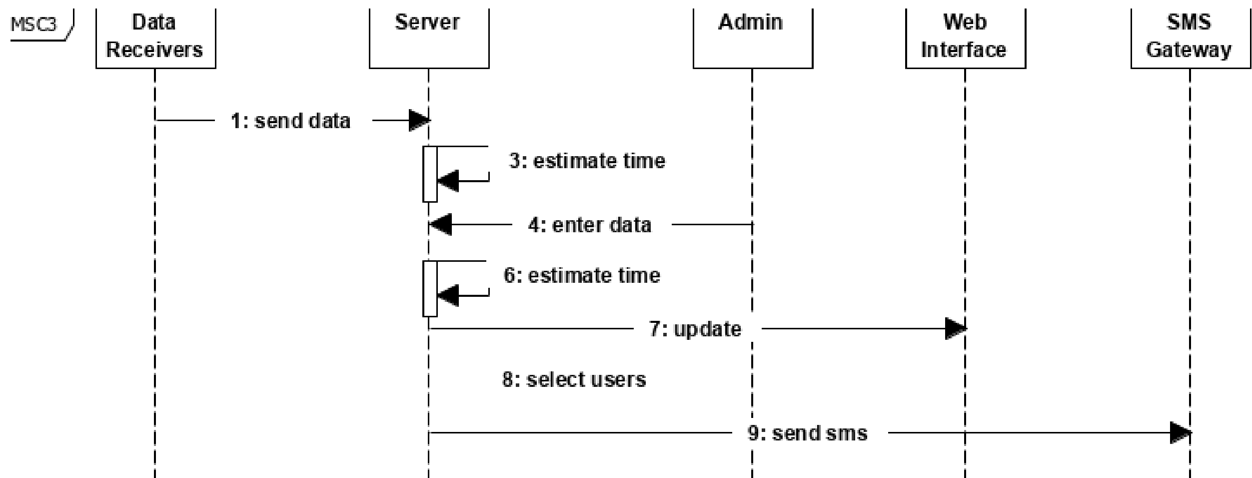

Figure 4.5: MSC3: Recalculate bus arrival according to manually entered data as illustrated in [1, Fig. 4]

4.3.2 Case Study 2: Boiler Control System

The boiler control system was described in [89] [2]. This system controls the temperature and the pressure of the boiler. It is modelled by four scenarios showing the interaction between its components. The scenarios, as well as the high-level structure, are presented in Figure 4.7. Using our approach, we detect two types of EBs, which are *NLBC* and *AC* as presented in Figure 4.8. The hMSC of the system branches into three scenarios *Analysis*, *Stop* and *Register*. The active processes in these scenarios are *Control*, *Control*, and *Sensor*, respectively. This situation may lead to *NLBC*. After the execution of the MSC *Register*, the system can choose between processing to the scenario *Register*, *Analysis* or *Stop*. As the active processes in the branches forming the fork are different, *Sensor* may choose the *Register* MSC, whereas Control may choose the *Analysis* MSC without synchronizing their choices.

Four asynchronous concatenation EBs were detected when we assume a synchronous concatenation between the scenarios of this case study. These EBs imply seven ISs and present changes in the order of the execution of the MSCs, which does not conform to the system specification. The component *Control* has interaction in the scenario *Initialise* but is a ready process in the scenario *Register*. The fact that *Control* is an active process in the

Results

********* Identical States *********
# EB_1: Shared States: The immediate state of process Server after its shared states belongs to the send events!
The following message can cause an emergent behavior:
Message Label: update
Sender: Server
Receiver: Web Interface
Scenario: MSC1
Local visual order: 5

======> The total number of shared states causing these EBs is 1

********* Race Condition (RC) **************
# EB_2 Race conditions (Case 1): (5)
- Receiver agent: Server
The change in the following received messages in MSC1 can change the process behavior
('GPS Data Rec?sendlocationdata!Server','Traffic Data Rec?sendtrafficdata!Server','Weather Data Rec?sendweatherdata!Server')
EB_3 Race conditions (Case 1): (5)
- Receiver agent: Server
The change in the following received messages in MSC2 can change the process behavior
('GPS Data Rec?sendlocationdata!Server','Traffic Data Rec?sendtrafficdata!Server','Weather Data Rec?sendweatherdata!Server')
EB_4 Race conditions (Case 1): (1)
- Receiver agent: Server
The change in the following received messages in MSC3 can change the process behavior
('Data Receivers?senddata!Server','Admin?enterdata!Server')

********* Non Local Branching Choice (NLBC) **************

********* Asynchronous Concatenation (AC) **************

Figure 4.6: The results of the analysis of the real-time fleet management system using our tool

successor scenarios *Analysis* to *Register* may results in two EBs (i.e., *Initialise* → *Analysis* and *Initialise* → *Stop*). Based on the system hMSC, we have the following feasible path: *Analysis* → *Register* → *Analysis*. However, another path, *Analysis* → *Analysis* may be implied because *Control* is an active process in *Analysis* whereas it is a ready process in *Register*. The EB *Analysis* → *Stop* is also caused by the component *Control* being in a ready state in *Register* and an active state in *Stop*. A consequence of the occurrence of the asynchronous concatenations is IS. For instance, the occurrence of the EB *Analysis* → *Stop* could result in the ISs *Stop* → *Analysis* and *Analysis*→ *Stop* → *Register*.

In the analysis of the system with the assumption of an asynchronous concatenation, another EBs was detected in addition to the EBs detected with the synchronous communication in the system (Figure 4.9). The component *Sensor* is an active component in *Register* which may lead to the system starting its execution by the *Register*. The IS resulting from this EB is *Register* → *Initialise*.

The last connection of the MSC register is identified as a non-local choice node (*Stop*-*Register*). In other words, it is a branching node through which directly related interactions

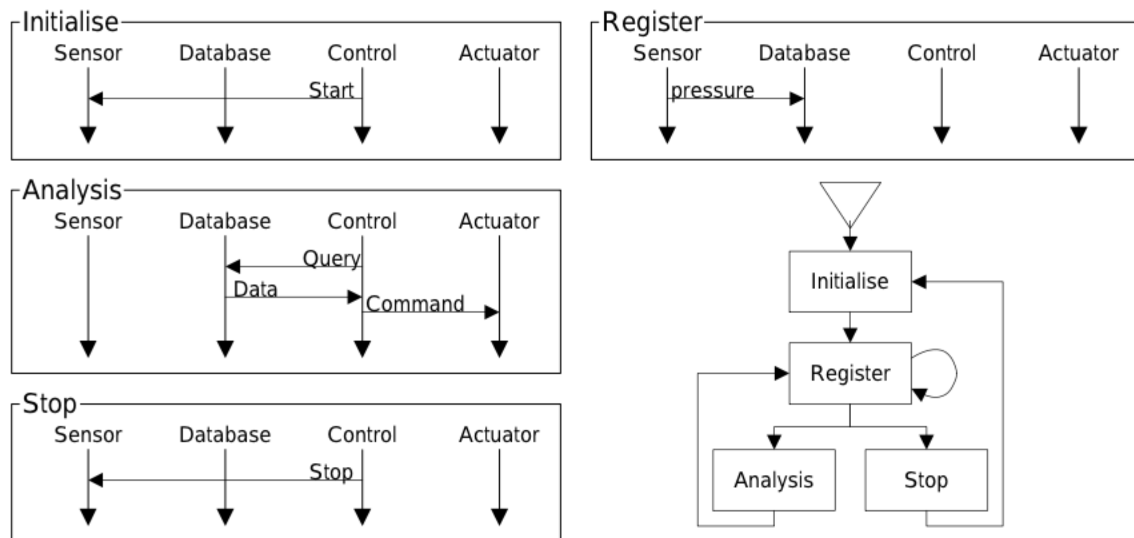

Figure 4.7: hMSC and MSCs presenting the boiler control system as illustrated in [2, Fig. 1]

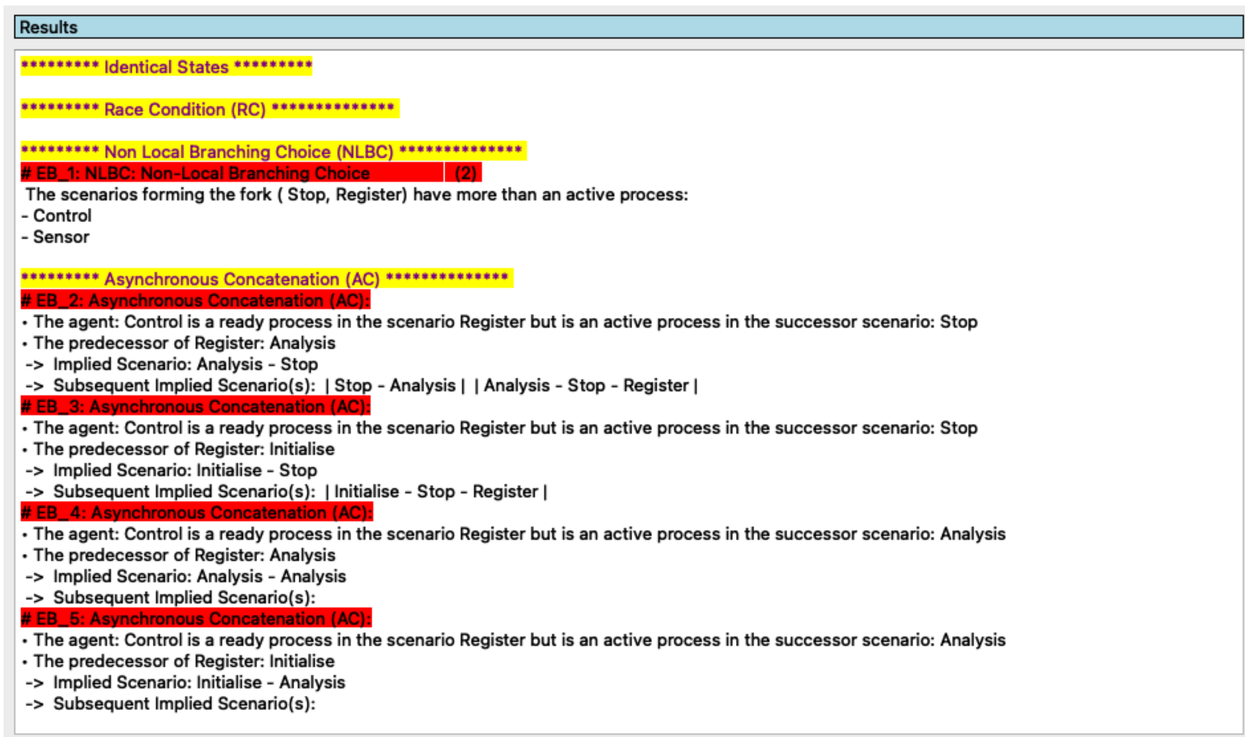

Figure 4.8: The result of the analysis of the boiler control system using our tool considering synchronous communication

Results

********* Identical States *********

********* Race Condition (RC) **************

********* Non Local Branching Choice (NLBC) **************
# EB_1: NLBC: Non-Local Branching Choice (2)
The scenarios forming the fork (Stop, Register) have more than an active process:
- Control
- Sensor

********* Asynchronous Concatenation (AC) **************
# EB_2: Asynchronous Concatenation (AC):
• The agent: Control is a ready process in the scenario Register but is an active process in the successor scenario: Stop
• The predecessor of Register: Analysis
-> Implied Scenario: Analysis - Stop
-> Subsequent Implied Scenario(s): | Stop - Analysis | | Analysis - Stop - Register |
EB_3: Asynchronous Concatenation (AC):
• The agent: Control is a ready process in the scenario Register but is an active process in the successor scenario: Stop
• The predecessor of Register: Initialise
-> Implied Scenario: Initialise - Stop
-> Subsequent Implied Scenario(s): | Initialise - Stop - Register |
EB_4: Asynchronous Concatenation (AC):
• The agent: Control is a ready process in the scenario Register but is an active process in the successor scenario: Analysis
• The predecessor of Register: Analysis
-> Implied Scenario: Analysis - Analysis
-> Subsequent Implied Scenario(s):
EB_5: Asynchronous Concatenation (AC):
• The agent: Control is a ready process in the scenario Register but is an active process in the successor scenario: Analysis
• The predecessor of Register: Initialise
-> Implied Scenario: Initialise - Analysis
-> Subsequent Implied Scenario(s):
EB_6: Asynchronous Concatenation (AC):
• The agent: Sensor is a ready process in the scenario Initialise but is an active process in the successor scenario: Register
• The predecessor of Initialise: Initial Node
-> Implied Scenario: Initial Node - Register
-> Subsequent Implied Scenario(s): | Initial Node - Register - Initialise |

Figure 4.9: The result of the analysis of the boiler control system using our tool considering asynchronous communication

Table 4.1: The number of scenarios and components in the different case studies

Case Studies	Number of processes	Number of scenarios	Number of interactions
ATM [90]	3	12	33
Boiler Control System [89]	4	4	6
Distributed Smart Camera System [56]	5	5	84
Fleet Management System [23]	8	4	68
Green House System 1 [82]	8	2	21
Green House System 2 [89]	3	2	8
GSM Mobility Management 1 [60]	4	14	64
GSM Mobility Management 2 [56]	4	24	94
MyPetStore [81]	5	5	7
Order Delivery System [91]	5	3	8
Real Time Fleet Management System [1]	8	3	60

are sent by different processes ($Control$-$Sensor$). At the fork level, the process control may start the execution of the scenario *Stop*, whereas the process *Sensor* starts the execution of the scenario *Register*.

4.3.3 Comparative Study

To provide different evaluation criteria of our methodology to the previous methodology, we selected the case studies to reflect different applications. Furthermore, the case studies, presented in Table 4.1, were selected from related research that adopted theoretical and practical work and is considered as the main works in the literature.

The results of the automatic program system analysis of the different use cases are reported in table 4.2. In the analysis and reporting of the number of EBs in the used cases, we assumed synchronous communication between the system's components in this section.

Our methodology focuses on the detection of EBs. The number of EBs detected could be significantly different from the number of ISs. For a proper comparison of our methodology to other methodologies, we opted for the estimation of the number of IS that may result from the occurrence of an EB. The calculation of the number of ISs is specific to the type of EB. This number also affected by the number of interactions and the number of scenarios affected by the EB.

For SS, identical states causing an SS may cause different ISs that are dependent other

Table 4.2: The number of EBs/ISs detected in each case study

Case Studies	SS	RDC	RC	NLBC	AC	Total EBs/ISs (our approach)	Total EBs/ISs (other approaches)
ATM	1(2)	0	0	1(2)	0	2(2)	2 EBs [23], 2 ISs [91]
Boiler Control System	0	0	0	1(2)	4(7)	5(9)	4 EBs [23], 7 ISs [60]
Distributed Smart Camera System	13	13	3	0	0	29	9 CBs [56]
Fleet Management System	7	1	4	4	3	19	1 EBs [23]
Green House System 1	1	2	0	0	2	5	1 EB [82]
Green House System 2	1	3	0	0	0	4	1 EB[89]
GSM Mobility Management 1	1(1)	3	0	2(4)	2(4)	8(12)	32 ISs [60]
GSM Mobility Management 2	10	3	0	0	4	17	16 CBs [56]
MyPetStore	0	0	0	1(2)	0	1(2)	1 EB [23], 3 ISs [81]
Order Delivery System	1(2)	0	0	0	2(4)	3(6)	1 EB[23], 4 ISs [60]
Real Time Fleet Management System	1	0	3	0	2(4)	3(6)	N/A

than to the number of occurrences of the identical states in different scenarios to the next interaction after the identical states. For RDC, the number of ISs is calculated based on the number of interactions in the identical states where either the sender or the receiver are not identical, and the number of scenarios in the identical states is detected. The race condition ISs are all the combinations that can occur on receiving the interactions that may cause the RC except the combinations defined in other scenarios.

For $NLBC$, the number of IS depends on the number of active processes in the fork causing the EB. However, the number of IS reported in AC refers to the infeasible path according to the hMSC of the system and not according to the MSC, like other types of EBs. However, if we consider the same calculation used for component level while calculation system level, the number of ISs increases. In this case, the number of IS also depends on the number of interactions in each scenario causing the EB.

The number of ISs resulting from EBs is presented between parentheses. Reporting the possible ISs is also needed to detect the occurrence of EBs in runtime. In runtime, the EBs may be detected in the form of IS.

The chart presented in Figure 4.10 gives information about the number of ISs, and the number of EBs detected in our work and in related work for each of the case studies presented in Table 4.2. It shows that our methodology detected more EBs and more ISs in the majority of case studies. In the case study GSM mobility management, the ISs detected by our work are lower than the number of ISs detected by related work. This may be explained by our different approach to calculate the number of ISs for the system level EBs.

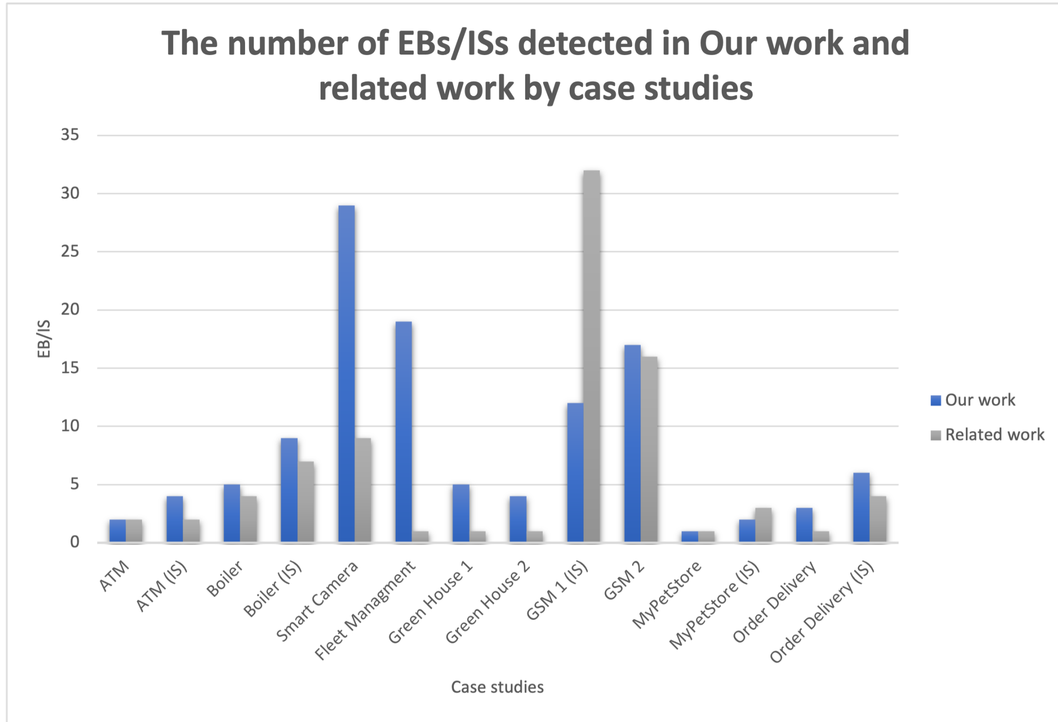

Figure 4.10: Number of EBs/ISs detected in our work versus related work for different case studies

Figure 4.11 presents the correlation between the number of detected EBs and the SBS data. The stronger the correlation is between the variable, the bigger the circles is. The figure shows a positive correlation, displayed in red, between the number of EBs detected and the number of interactions within the SBS. However, a negative correlation, displayed in blue, is displayed between the number of processes and EBs detected.

4.4 Dynamic Verification

In this section, we analyze the dynamic behaviour of the case studies to check if the system behaviour matches the behaviour specified in SBS or shows unexpected behaviours as detected by our methodology. The software behaviour is verified by the simulation of the software execution based on SBS. The collected traces are analyzed using statistical modelling and data mining approaches.

To simulate the case studies presented in sections 4.3.1 and 4.3.2, we implemented SBS

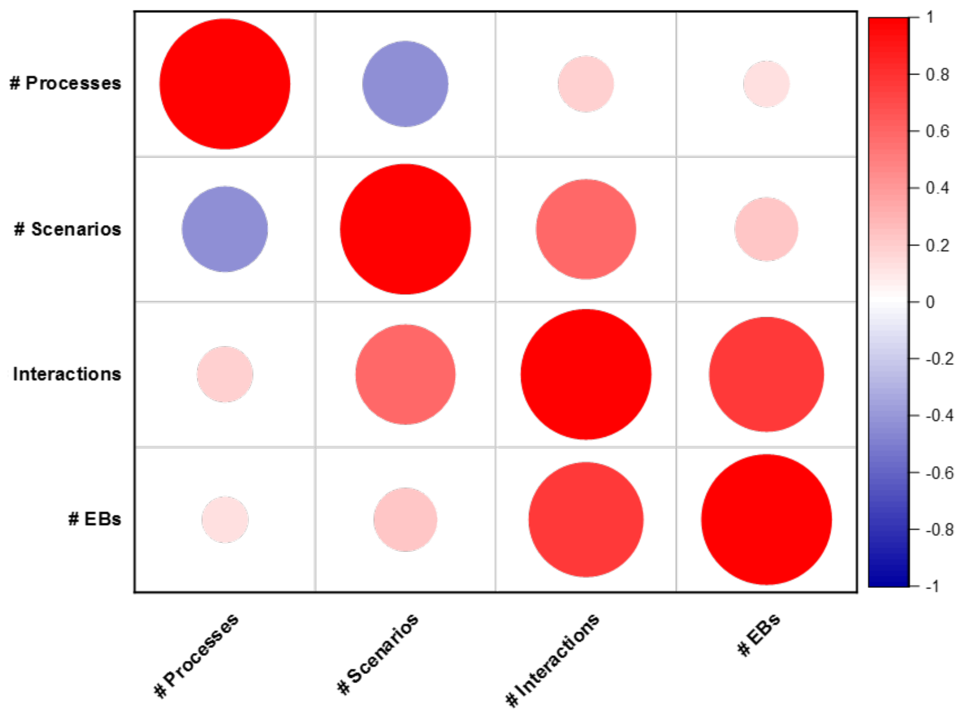

Figure 4.11: Correlation between the number of EBs detected and the SBS

of the system using JADE, which presents an efficient environment for distributed agent systems.

The traces of execution of the system could be different in every runtime which lead as to the collection of system traces for several executions. In the following, we are detailing only two case studies.

4.4.1 Statistical Modelling

Statistical modelling is used to model the outcome of comparison of the hypotheses given empirical data. The empirical data is the traces collected by the simulation of SBS. Each hypothesis corresponds to Markov Chain Model presenting the interactions within the system. The synthetic dataset used to generate an order ranking of the hypotheses considering the system's traces is generated through a random walk through the Markov Chain. The system's simulation supposes that the system's behaviour conforms to its specifications. However, according to our analysis, the system may show EBs under certain circumstances. Thus, we compare the results obtained by applying our methodology to those assuming that the transitions between the runtime scenarios are always compliant to its hMSC. Each hypothesis is illustrated in a distinct colour. The belief level in a given hypothesis corresponds to the hypothesis weighting factor k, which is reported in the x-axis. The evidence is presented in the y-axis. For the same values of k, stronger evidence means a higher plausibility.

Case Study 1: Real-Time Fleet Management System

In this section, we analyze the behaviour of the system by the analysis of the order of execution of the interactions within the scenarios. Thus, we considered the global order of the interaction as an atomic unit. In this section, we are studying the data collected by comparing it to the hypothesis. Each hypothesis expresses how the data could be produced. The first hypothesis (H1) assumes the transition between one interaction to the next according to scenario-based specification, the second hypothesis (H2) assumes that unexpected

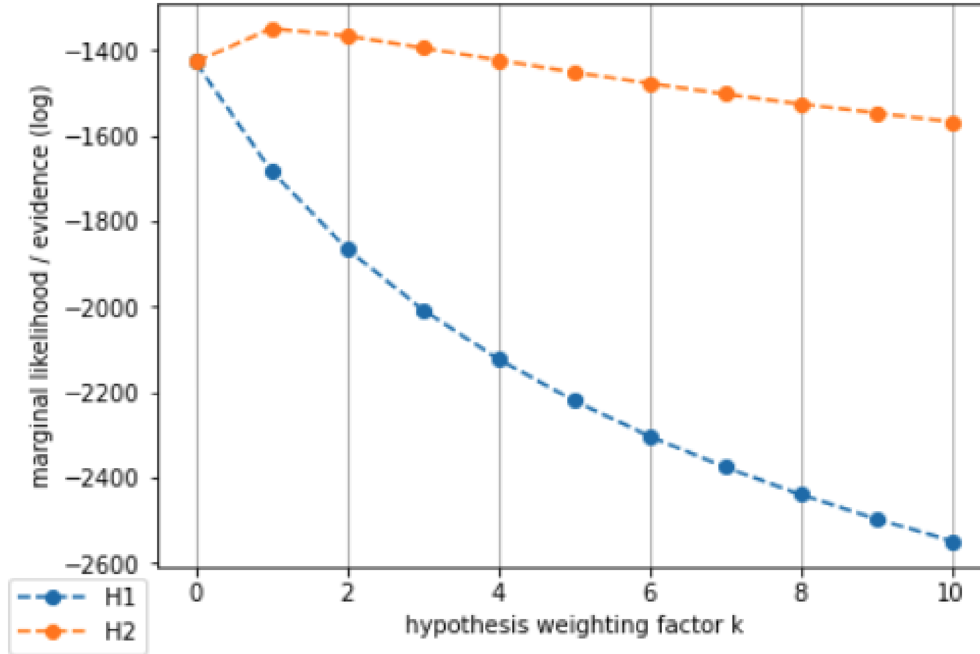

Figure 4.12: Case study 1: Results of comparing hypothesis using Hyptrails

behaviours are depicted by our analysis of the system in section 4.3.1 occurred during the system execution. The results of our analysis are presented in Figure 4.12. The results show that our hypothesis assuming the occurrence of EBs in runtime is more plausible than the hypothesis assuming the execution of the system according to SBS. For a more detailed analysis of the type of EBs that occurred, we opt to analyze the system trace using sequential pattern mining analysis.

Case Study 2: Boiler Control System

The set of hypotheses considered in this analysis are as follows: (H1) the system behaviour matches SBS, (H2) the system behaves according to the system specification while showing EBs at times, (H3) the system may re-execute the same scenario infinitely in the presence of a loop in the hMSC, (H4) the system keeps re-executing the same scenarios. We model each hypothesis in the form of a graph. The generated directed graphs representing the hypotheses are shown in Figure 4.13.

The graph 4.14, presents the results of the comparison of the four hypotheses. The EB

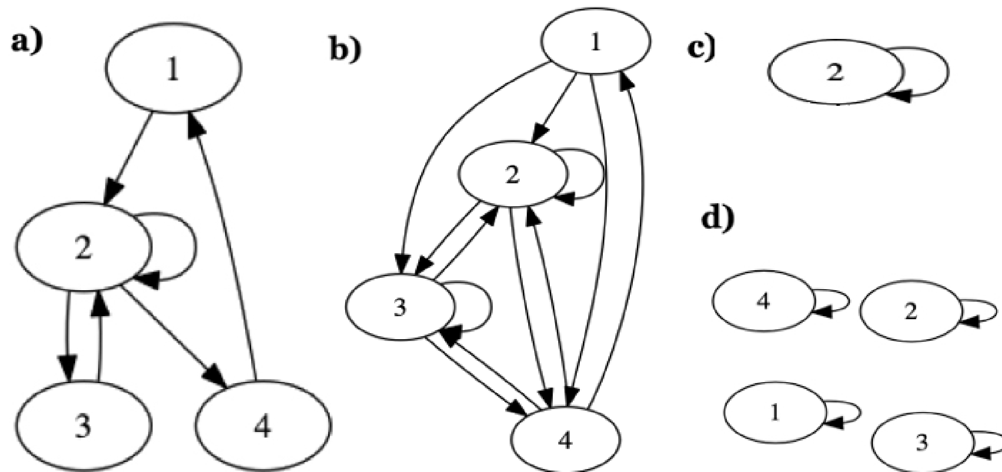

Figure 4.13: Network graphs presentation of H1 in (a), H2 in (b), H3 in (c) and H4 in (d)

hypothesis is the most plausible one.

4.4.2 Sequential Pattern-Mining Analysis

Using sequential pattern mining analysis, we analyze the system traces which are delivered in a sequence to find relevant patterns. In this research, we are concerned about the relation between the system component to reveal component level EBs, and we are concerned about the relation between the scenarios to reveal system-level EBs.

We analyzed two case studies to study the behaviour of the system. The real-time fleet management system is used to study the behaviour of the system at a component level, whereas the boiler control system at a system level. This analysis allows us to confirm the validity of our approach and reveal the system's behaviour at runtime.

Case Study 1: Real-Time Fleet Management System

In this analysis, we focused on detecting the system's behaviour according to the local order of the interaction within the scenarios. According to our hypothesis, the system behaviour at runtime may differ from the system behaviour according to the scenario-based specification. To reveal the sequential pattern of the messages exchanged between the components, we

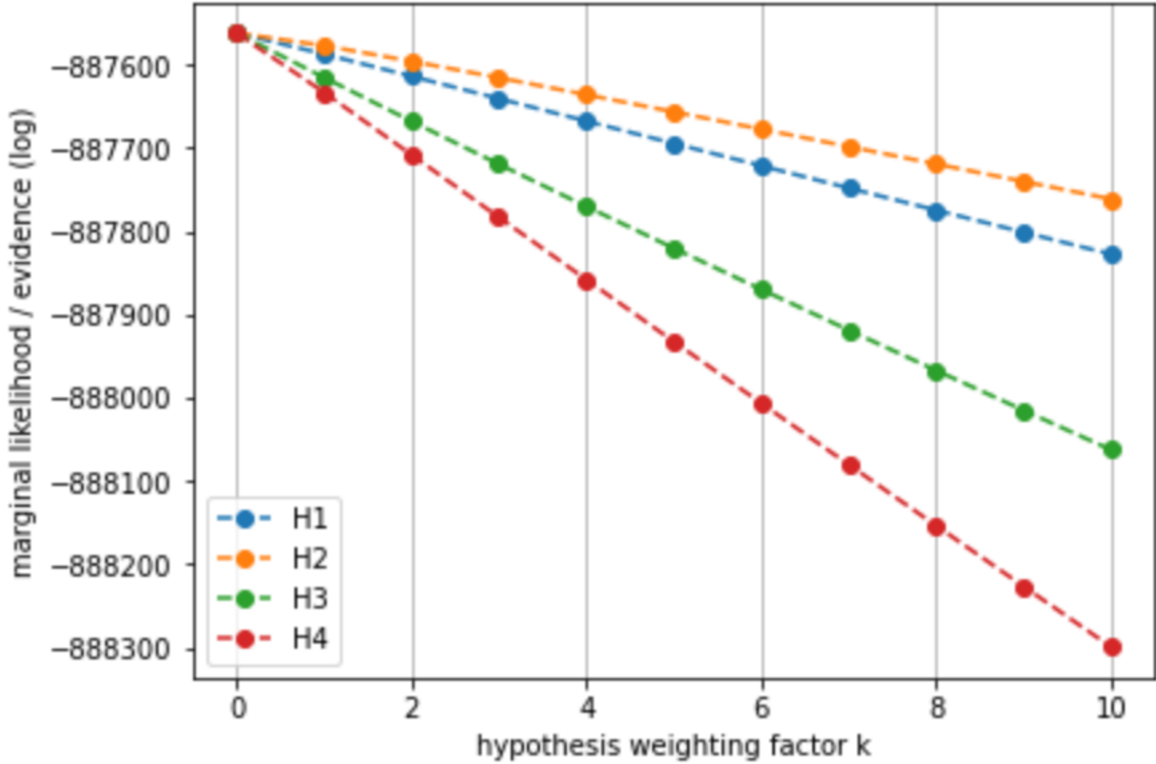

Figure 4.14: Case study 2: Results of comparing hypothesis using Hyptrails

opted to compare the system's traces at different runtime. Each trace is the recording of the message interaction within the system within a fixed lapse of time. According to our analysis in section 4.3.1, a *SS* was detected. To catch this behaviour at runtime, the atomic unit considered in this analysis is the word composed of the message label, the local order and the scenario ID. The results of the contiguous pattern mining of the traces reveal twenty-nine frequent patterns with different support. The modelling of the patterns having the maximum support shows the unexpected behaviours. The IS shown in Figure 4.15, 4.16, 4.17 are presenting the ISs detected in the traces and caused by a race condition based on our analysis, whereas the ISs caused by shared states are shown in Figure 4.18.

Case Study 2: Boiler Control System

To show the frequency of transitions between the scenarios, we convert the patterns' support, resulting in the analysis of the system's tracing to a state transition probability matrix of a

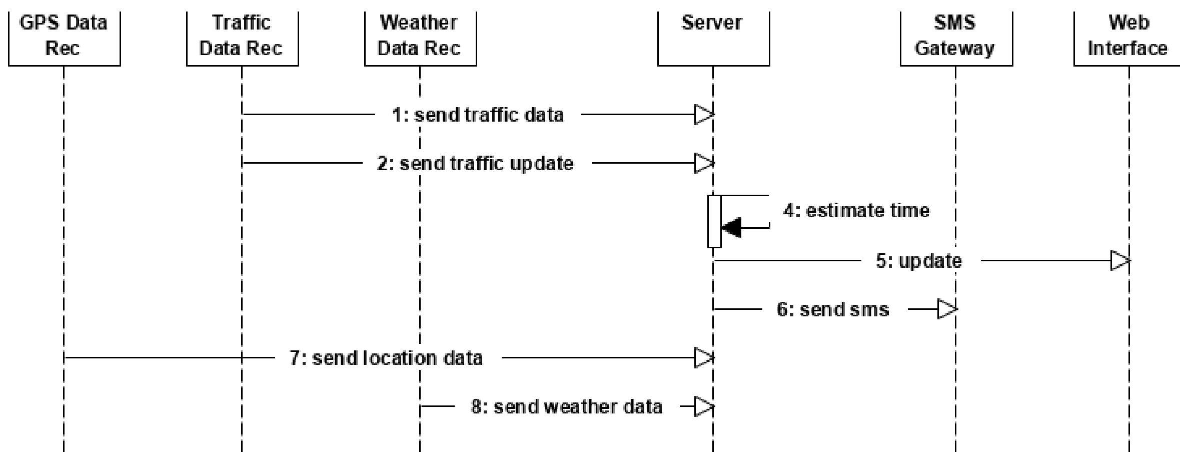

Figure 4.15: RC ISs detected in real-time fleet management system

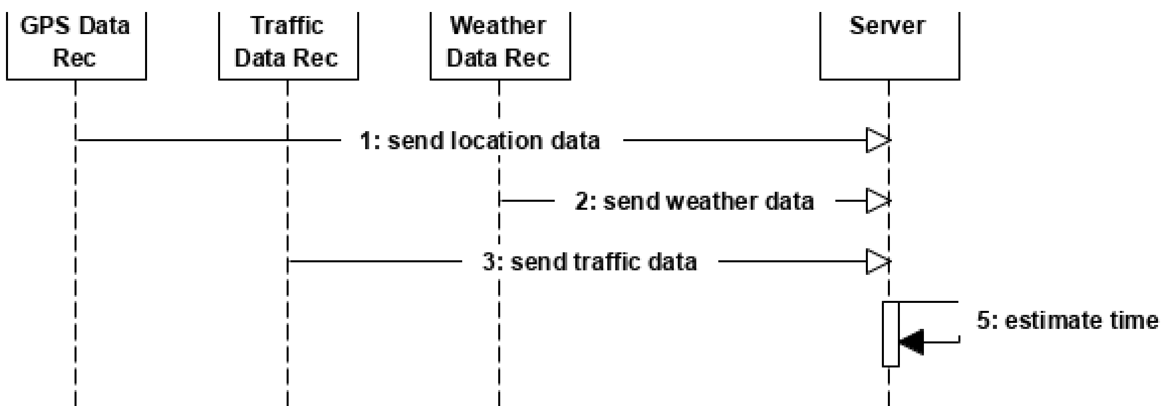

Figure 4.16: RC ISs detected in real-time fleet management system

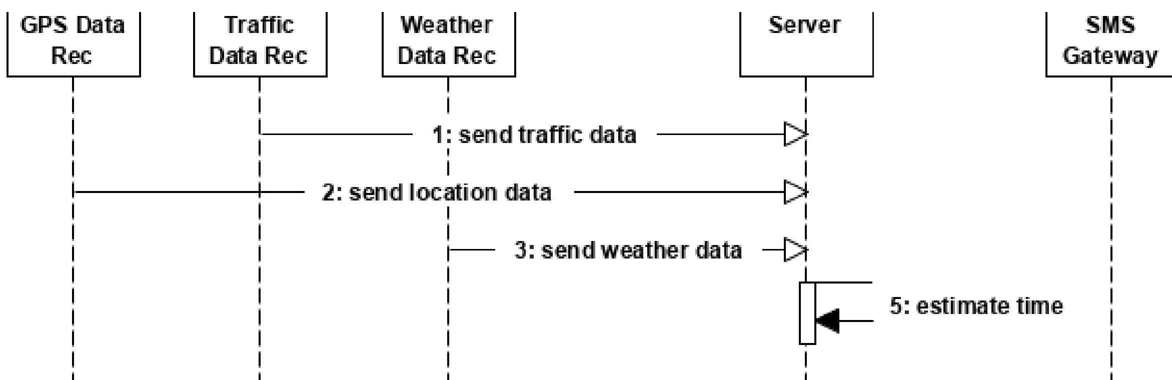

Figure 4.17: The RC ISs detected in real-time fleet management system

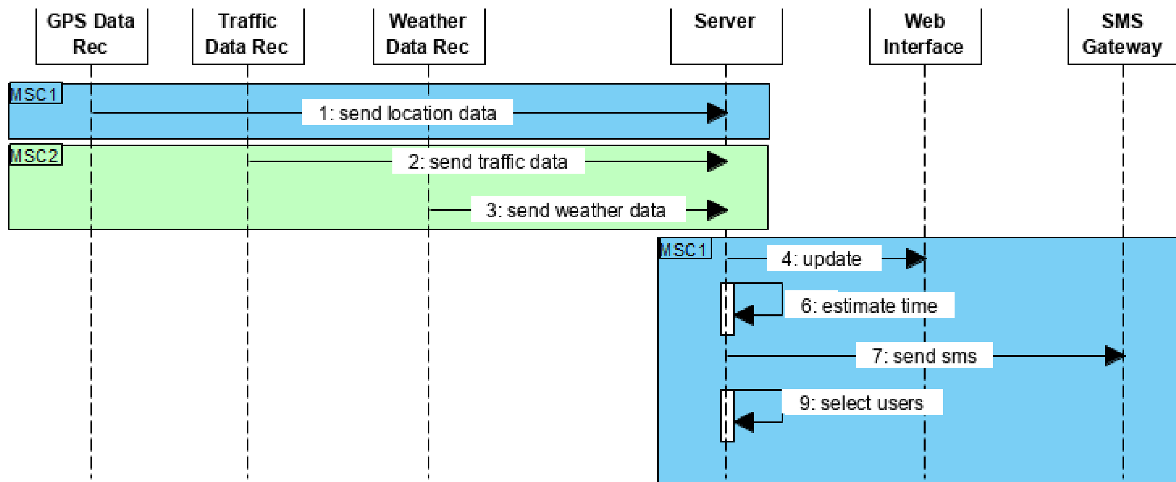

Figure 4.18: SS ISs detected in real-time fleet management system

Markov chain. As shown in Figure 4.19, this matrix presents the probability of EBs is less than the probability of the system's behaviours defined in the scenario-based specification.

The number of the pattern corresponding to the EBs are less than the legal path. This supports the hypothesis that EBs may or may not occurs in runtime. The analysis of scenario specification in early phases allows the consideration of these special cases not only in the implementation of the system but also in the testing. This also allows the definition of special test cases for the detection of this rare failure.

Based on our analysis of Boiler Control System scenarios, *Sensor* may initiate the system execution as it is an active process in *Register*, whereas it is in a ready state at the scenario *Initialise*. The analysis of the collected traces of different runtimes shows that the *Sensor* initiates the system execution in *Register* at a rate of 50

4.5 Threats to Validity

This section will discuss the threats to the validity of our findings in the presented case studies. As external validity, we cite that there are no specific benchmarks for the evaluation of the detection of EBs. However, the evaluation is performed by the application of the proposed methodology on case studies selected from the main works of this field. The internal

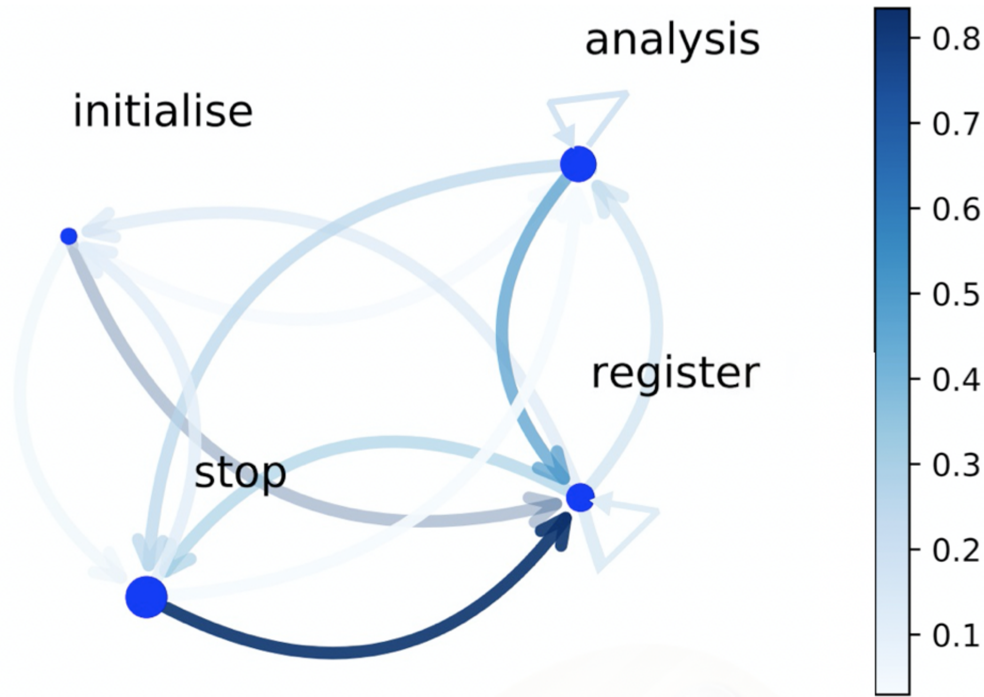

Figure 4.19: Network graph presenting the behaviour of the Boiler System using sequential pattern mining algorithm

validity is the simulation of the case studies by the author. The simulation of the system was required to collect the message exchange between the different components of the system. Furthermore, the complexity of the code may thread the collection of the data. Thus, it requires the expertise of the system developers. However, the simulation was faithful to the case studies scenario-specification modelling and avoided the impact of other decisions made in later phases of the software development, which can compromise the system's behaviour. As this methodology is a non-message content methodology, monitoring a real-time system will not be a promising technique to evaluate this methodology.

4.6 Conclusion

This chapter describes the detection of different types of EBs on case studies based on system modelling. The automated tool assists in the verification processes to check for weaknesses in the models that may lead to unwanted behaviours. We also present a dynamic analysis

of the case studies to verify our methodology's accuracy, which supports our work.

Chapter 5

Conclusion and Future Work

Given the enormous and distributed scale of software systems today, it is challenging to create complete and quality system specifications. However, a poor specification may lead to critical bugs. This research's primary goal is to highlight the system modelling issues to prevent errors during the execution time. Thus we focused on detecting patterns in SBS that could be the cause of unwanted behaviours. In this research, we opted to analyze the SBS as the early detection of failures may significantly decrease the time and the cost of the software. The proposed methodology is supported by case studies and the analysis of the system's simulation traces. This analysis is primarily used to prove the validity and the correctness of our analysis but can also be used to depict the behaviour of the system at runtime.

The contribution of this research is the direct detection of EBs from SBS using data-mining techniques. Furthermore, the automatic analysis of SBS offers a guide for developers to implement better-distributed software systems.

Therefore, this work addresses analyzing SBS for the detection of known EBs using a homogeneous and automated methodology. The main contributions of this research are as follow:

- Automatic and homogeneous methodology for the detection of different types of EBs

in DSS.

- Presentations of the results in a textual and graphical format to facilitate the detection of EBs.

- Experimental case studies to evaluate the performance of the algorithm.

- Analyses of runtime information of a system to verify its behavioural aspects.

The proposed methodology is not domain-specific and is only dependent on SBS. SBS, such as UML and MSC, are more mastered by engineers than the formal languages which are challenging and costly to adopt by untrained engineers. This methodology allows the automatic identification of patterns in SBS that may lead to EBs. This methodology is specific to DSS and MAS.

During the implementation of a complex system, the manual analysis gets too complicated for the designer to implement the right product. Our approach supports the developers in reducing probable failures before building the system and testing it. Besides increasing the level of automation, the implemented tool assures the early detection of errors which is cost-effective in software development.

The evaluation of practical works in previous related research is performed by comparing the case studies from the literature. In this work, besides adopting the same evaluation method, we simulated the case studies to prove the existence of the detected EBs using our methodology in runtime.

The limitation of this proposed methodology is being only suitable for systems lacking central control and composed of multiple components interacting with each other. Moreover, in this work, we applied our methodology in simulated systems instead of real systems. The simulation of the system ensures that the system is implemented according to the SBS, which is required to prove the correctness of the proposed methodology. Furthermore, reverse engineering the code of DSS is a complex process in the absence of real system specification. The analysis results of the simulation results gave confidence to the provided methodology.

Model and predict software failure behaviour based on the software system logs is one of the directions that can be followed to complete this research. Furthermore, the use of SBS for testing purposes would be efficient in preventing undesirable behaviours of the system during its execution. Thus, building a framework for testing DSS and MAS is one of the future works. This requires the investigation of the automated test case generation according to the type of EBs. This work can also be extended by building a tool to monitor the behaviour of a DSS in runtime using the results obtained by analyzing the modelling based on our methodology.

Another research area that can be completed in the future is the generation of model-based specifications from software requirements. Software requirements written in informal specification languages make the communication between the user and developer easy, which this aids in understanding the problem and can be beneficial in the generation of software design. The generated requirements can then be mapped for the generation of scenario specification models. Thus, studying the conversion of requirements to models is needed. The generated model could be used in this work to detect EBs.

Bibliography

[1] B. Khurshid, M. Moshirpour, A. Eberlein, and B. H. Far, "An automated ontology generation technique for an emergent behavior detection system," in *IEEE 14th International Conference on Information Reuse & Integration, IRI 2013, San Francisco, CA, USA, August 14-16, 2013*. IEEE Computer Society, 2013, pp. 380–387. [Online]. Available: https://doi.org/10.1109/IRI.2013.6642496

[2] S. Uchitel, J. Kramer, and J. Magee, "Implied scenario detection in the presence of behaviour constraints," *Electronic Notes in Theoretical Computer Science*, vol. 65, no. 7, pp. 65–84, 2002.

[3] K. Iwanicki, "A distributed systems perspective on industrial iot," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1164–1170.

[4] R. Nasim and A. Kassler, "Distributed architectures for intelligent transport systems: A survey," in *2012 Second Symposium on Network Cloud Computing and Applications*, 12 2012, pp. 130–136.

[5] M. van Steen and A. S. Tanenbaum, "A brief introduction to distributed systems," *Computing*, vol. 98, no. 10, pp. 967–1009, 2016.

[6] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *Industrial Informatics, IEEE Transactions on*, vol. 9, pp. 1234–1249, 08 2013.

[7] M. Moshirpour, A. Mousavi, and B. H. Far, "Detecting Emergent Behavior In Distributed Systems Using Scenario-Based Specifications," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 06, pp. 729–746, 2012. [Online]. Available: https://doi.org/10.1142/S0218194012400104

[8] A. Mousavi, "Inference of emergent behaviours of scenario-based specifications," 2009. [Online]. Available: https://prism.ucalgary.ca/handle/1880/103708

[9] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, "Verdi: A framework for implementing and formally verifying distributed systems," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 357–368. [Online]. Available: https://doi.org/10.1145/2737924.2737958

[10] T. G. Grbac, P. Runeson, and D. Huljenić, "A quantitative analysis of the unit verification perspective on fault distributions in complex software systems: an operational replication," *Software quality journal*, vol. 24, no. 4, pp. 967–995, 2016.

[11] M. Moshirpour, "Model-based Analysis of Software Requirements for Distributed Software Systems," 2016.

[12] J. Whittle and J. Schumann, "Scenario-Based Engineering of Multi-Agent Systems," in *Agent Technology from a Formal Perspective*, ser. NASA Monographs in Systems and Software Engineering. London: Springer London, 2006, pp. 159–189.

[13] K. Krogmann, *Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis*. KIT Scientific Publishing, 2012.

[14] Y. Li, T. Tan, and J. Xue, "Understanding and analyzing java reflection," *ACM Trans. Softw. Eng. Methodol.*, vol. 28, no. 2, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3295739

[15] H. Östlund, "JRA: offline analysis of runtime behaviour," in *Companion to the 19th annual ACM SIGPLAN conference on object-oriented programming systems, languages, and applications*, ser. OOPSLA '04. ACM, 2004, pp. 16–17.

[16] C. Xie, W. Jeong, G. Matyasfalvi, H. Van Dam, K. Mueller, S. Yoo, and W. Xu, "Exploratory Visual Analysis of Anomalous Runtime Behavior in Streaming High Performance Computing Applications," in *Computational Science – ICCS 2019*, J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, and P. M. A. Sloot, Eds. Cham: Springer International Publishing, 2019, pp. 153–167.

[17] L. Addazi and F. Ciccozzi, "Blended graphical and textual modelling for UML profiles: A proof-of-concept implementation and experiment," *Journal of Systems and Software*, vol. 175, p. 110912, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121221000091

[18] M. Murtazina and T. V. Avdeenko, "An Ontology-based Approach to Support for Requirements Traceability in Agile Development," *Procedia Computer Science*, vol. 150, pp. 628–635, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919303898

[19] J. Sun and J. S. Dong, "Synthesis of Distributed Processes from Scenario-Based Specifications," in *FM 2005: Formal Methods*, J. Fitzgerald, I. J. Hayes, and A. Tarlecki, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 415–431.

[20] K. S. Mishra and A. K. Tripathi, "Some issues, challenges and problems of distributed software system," *International Journal of Computer Science and Information Technologies. Varanasi, India*, vol. 7, no. 3, 2014.

[21] A. Khemiri, M. E. A. Hamri, C. Frydman, and J. Pinaton, "Limiting State Space Explosion of Model Checking Using Discrete Event Simulation: Combining DEVS and

PROMELA," in *Proceedings of the 2019 Summer Simulation Conference*, ser. SummerSim '19. San Diego, CA, USA: Society for Computer Simulation International, 2019.

[22] E. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model Checking and the State Explosion Problem," in *Tools for Practical Software Verification.* Springer, Berlin, Heidelberg, jan 2012, pp. 1–30.

[23] F. H. Fard, "Detecting and fixing emergent behaviors in distributed software systems using a message content independent method," Ph.D. dissertation, University of Calgary, 2016. [Online]. Available: http://ezproxy.lib.ucalgary.ca/login?url=https://search-proquest-com.ezproxy.lib.ucalgary.ca/docview/1923116789?accountid=9838

[24] Z. Jin, "Chapter 2 - Requirements Engineering Methodologies," in *Environment Modeling-Based Requirements Engineering for Software Intensive Systems*, Z. Jin, Ed. Oxford: Morgan Kaufmann, 2018, pp. 13–27. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128019542000029

[25] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software*, vol. 172, p. 110851, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220302417

[26] D. Harel, "From Play-In Scenarios to Code: An Achievable Dream," in *Fundamental Approaches to Software Engineering*, T. Maibaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 22–34.

[27] S. Uchitel, G. Brunet, and M. Chechik, "Behaviour model synthesis from properties and scenarios," in *29th International Conference on Software Engineering (ICSE'07)*, 2007, pp. 34–43.

[28] Z. Fang, H. Fu, T. Gu, Z. Qian, T. Jaeger, P. Hu, and P. Mohapatra, "A Model Checking-Based Security Analysis Framework for IoT Systems," *High-Confidence*

Computing, p. 100004, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295221000052

[29] I. T. U. T. S. S. ITU-T, "Z.120 Message Sequence Chart (Edition 5.0)," *Language*, p. 146, 2011.

[30] J. C. Mogul, "Emergent (mis)behavior vs. complex software systems," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, p. 293–304, Apr. 2006. [Online]. Available: https://doi.org/10.1145/1218063.1217964

[31] C.-C. Chen, S. B. Nagl, and C. D. Clack, "Specifying, detecting and analysing emergent behaviours in multi-level agent-based simulations," in *Proceedings of the 2007 Summer Computer Simulation Conference*, ser. SCSC '07. San Diego, CA, USA: Society for Computer Simulation International, 2007, p. 969–976.

[32] C. M. Belcastro, *Validation and Verification Techniques and Tools.* London: Springer London, 2013, pp. 1–8. [Online]. Available: https://doi.org/10.1007/978-1-4471-5102-9{_}146-1

[33] K. Giammarco and K. Giles, "Verification and Validation of Behavior Models Using Lightweight Formal Methods BT," in *Disciplinary Convergence in Systems Engineering Research*, A. M. Madni, B. Boehm, R. G. Ghanem, D. Erwin, and M. J. Wheaton, Eds. Cham: Springer International Publishing, 2018, pp. 431–447.

[34] M. Moshirpour, N. Mani, A. Eberlein, and B. H. Far, "Model based approach to detect emergent behavior in multi-agent systems," *12th International Conference on Autonomous Agents and Multiagent Systems 2013, AAMAS 2013*, vol. 2, no. Aamas, pp. 1285–1286, 2013. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84899438919{&}partnerID=40{&}md5=7e0c168e8cc9a95a9cc51b9e45d54af4

[35] J. Krause, E. Hintze, S. Magnus, and C. Diedrich, "Model Based Specification, Verification, and Test Generation for a Safety Fieldbus Profile," in *Computer Safety, Reliability,*

and Security, F. Ortmeier and P. Daniel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 87–98.

[36] E. del Val, C. Martínez, and V. Botti, "A Multi-agent Framework for the Analysis of Users Behavior over Time in On-Line Social Networks," in *10th International Conference on Soft Computing Models in Industrial and Environmental Applications*, Á. Herrero, J. Sedano, B. Baruque, H. Quintián, and E. Corchado, Eds. Cham: Springer International Publishing, 2015, pp. 191–201.

[37] T. Jussila, J. Dubrovin, T. Junttila, T. Latvala, I. Porres, and J. K. U. Linz, "Model checking dynamic and hierarchical uml state machines," *Proc. MoDeV2a: Model Development, Validation and Verification*, pp. 94–110, 2006.

[38] D. Marcilio, C. A. Furia, R. Bonifácio, and G. Pinto, "SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings," *Journal of Systems and Software*, vol. 168, p. 110671, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412122030128X

[39] A. Ribeiro, P. Meirelles, N. Lago, F. Kon, A. R. B, P. Meirelles, N. Lago, and F. Kon, "Ranking Source Code Static Analysis Warnings for Continuous Monitoring of FLOSS Repositories," in *Open Source Systems: Enterprise Software and Solutions*, I. Stamelos, J. M. Gonzalez-Barahoña, I. Varlamis, and D. Anagnostopoulos, Eds., vol. 1. Cham: Springer International Publishing, 2018, pp. 90–101.

[40] A. Habib and M. Pradel, "How many of all bugs do we find? a study of static bug detectors," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018, pp. 317–328.

[41] A. Lux and A. Starostin, "A tool for static detection of timing channels in Java," *Journal of Cryptographic Engineering*, vol. 1, no. 4, pp. 303–313, 2011.

[42] M. Weißbrich, L. Gerlach, H. Blume, A. Najafi, A. García-Ortiz, and G. Payá-Vayá, "FLINT+: A runtime-configurable emulation-based stochastic timing analysis framework," *Integration*, vol. 69, pp. 120–137, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926017307794

[43] D. F. Mendonça, G. N. Rodrigues, R. Ali, V. Alves, and L. Baresi, "GODA: A goal-oriented requirements engineering framework for runtime dependability analysis," *Information and Software Technology*, vol. 80, pp. 245–264, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584916301471

[44] L.-O. Damm, "Early and Cost-Effective Software Fault Detection - Measurement and Implementation in an Industrial Setting," Ph.D. dissertation, Blekinge Institute of Technology, 2007. [Online]. Available: http://www.bth.se/fou/forskinfo.nsf/6753b78eb2944e0ac1256608004f0535/5640e4544893e7e6c12572bb003cf89c?OpenDocument

[45] W. Khan, H. Ullah, A. Ahmad, K. Sultan, A. J. Alzahrani, S. D. Khan, M. Alhumaid, and S. Abdulaziz, "CrashSafe: a formal model for proving crash-safety of Android applications," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, p. 21, 2018. [Online]. Available: https://doi.org/10.1186/s13673-018-0144-7

[46] A. Mokhov, G. Lukyanov, and J. Lechner, "Formal Verification of Spacecraft Control Programs (Experience Report)," in *Proceedings of the 12th ACM SIGPLAN International Symposium on Haskell*, ser. Haskell 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 139–145. [Online]. Available: https://doi.org/10.1145/3331545.3342593

[47] M. E. Akintunde, E. Botoeva, P. Kouvaros, and A. Lomuscio, "Formal verification of neural agents in non-deterministic environments," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '20.

Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, p. 25–33.

[48] M. Hachicha, R. B. Halima, and A. H. Kacem, "Formal Verification approaches of Self-adaptive Systems: A Survey," *Procedia Computer Science*, vol. 159, pp. 1853–1862, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050919315583

[49] G. Capobianco, U. Di Giacomo, F. Mercaldo, and A. Santone, "Dunuen: A user-friendly formal verification tool," *Procedia Computer Science*, vol. 159, pp. 1431–1438, 01 2019.

[50] A. Souri, A. M. Rahmani, N. J. Navimipour, and R. Rezaei, "A symbolic model checking approach in formal verification of distributed systems," *Human-centric Computing and Information Sciences*, vol. 9, no. 1, p. 4, 2019. [Online]. Available: https://doi.org/10.1186/s13673-019-0165-x

[51] H. Günther and G. Weissenbacher, "Incremental Bounded Software Model Checking," in *Proceedings of the 2014 International SPIN Symposium on Model Checking of Software*, ser. SPIN 2014. New York, NY, USA: Association for Computing Machinery, 2014, pp. 40–47. [Online]. Available: https://doi.org/10.1145/2632362.2632374

[52] S. Merz, *Formal Specification and Verification.* New York, NY, USA: Association for Computing Machinery, 2019, p. 103–129. [Online]. Available: https://doi.org/10.1145/3335772.3335780

[53] L. Birdsey and C. Szabo, "An architecture for identifying emergent behavior in multi-agent systems," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, p. 1455–1456.

[54] S. Uchitel, J. Kramer, and J. Magee, "Detecting implied scenarios in message sequence

chart specifications," *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, vol. 26, 07 2001.

[55] A. Slama, F. H. Fard, and B. Far, "A Hybrid System for Detection of Implied Scenarios in Distributed Software Systems (S)," *Proceedings of the 30th International Conference on Software Engineering and Knowledge Engineering*, vol. 2018, no. June, pp. 360–396, 2018.

[56] C. B. de Melo, A. L. F. Cançado, and G. N. Rodrigues, "Characterization of implied scenarios as families of common behavior," *Journal of Systems and Software*, vol. 158, no. September, 2019.

[57] R. Alur, K. Etessami, and M. Yannakakis, "Inference of message sequence charts," in *Proceedings of the 22nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 304–313. [Online]. Available: https://doi.org/10.1145/337180.337215

[58] H. Muccini, "Detecting implied scenarios analyzing non-local branching choices," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2621, pp. 372–386, 2003.

[59] I. Song, S. Jeon, and D. Bae, "A graph based approach to detecting causes of implied scenarios under the asynchronous and synchronous communication styles," in *2009 16th Asia-Pacific Software Engineering Conference*, 2009, pp. 53–60.

[60] I. G. Song, S. U. Jeon, A. R. Han, and D. H. Bae, "An approach to identifying causes of implied scenarios using unenforceable orders," *Information and Software Technology*, vol. 53, no. 6, pp. 666–681, 2011. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2010.11.007

[61] N. A. K. Desai and A. Ganatra, "Efficient constraint-based Sequential Pattern Mining

(SPM) algorithm to understand customers' buying behaviour from time stamp-based sequence dataset," *Cogent engineering*, vol. 2, no. 1, 2015.

[62] M. Zihayat, H. Davoudi, and A. An, "Mining significant high utility gene regulation sequential patterns," *BMC systems biology*, vol. 11, no. Suppl 6, p. 109, 2017.

[63] X. Yuan, W. Chang, S. Zhou, and Y. Cheng, "Sequential Pattern Mining Algorithm Based on Text Data: Taking the Fault Text Records as an Example," *Sustainability (Basel, Switzerland)*, vol. 10, no. 11, p. 4330, 2018.

[64] D. Liu, D. Liu, S. Cai, S. Cai, X. Guo, and X. Guo, "Incremental sequential pattern mining algorithms of Web site access in grid structure database," *Neural computing & applications*, vol. 28, no. 3, pp. 575–583, 2017.

[65] Y. Abboud, A. Brun, and A. Boyer, "C3Ro: An efficient mining algorithm of extended-closed contiguous robust sequential patterns in noisy data," *Expert Systems with Applications*, vol. 131, pp. 172–189, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741741930288X

[66] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements bt," in *Advances in Database Technology — EDBT '96*, P. Apers, M. Bouzeghoub, and G. Gardarin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 1–17.

[67] Jian Pei, Jiawei Han, B. Mortazavi-Asl, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu, "Prefixspan,: mining sequential patterns efficiently by prefix-projected pattern growth," in *Proceedings 17th International Conference on Data Engineering*, 2001, pp. 215–224.

[68] R. V. Mane, "A Comparative Study of Spam and PrefixSpan Sequential Pattern Mining Algorithm for Protein Sequences," in *Advances in Computing, Communication, and*

Control, S. Unnikrishnan, S. Surve, and D. Bhoir, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 147–155.

[69] P. Fournier Viger, C.-W. Lin, U. Rage, Y. S. Koh, and R. Thomas, "A Survey of Sequential Pattern Mining," *Data Science and Pattern Recognition*, vol. 1, pp. 54–77, 2017.

[70] P. Fournier Viger, *Fast Vertical Sequential Pattern Mining Using Co-occurrence Information.*, may 2014.

[71] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information," in *Advances in Knowledge Discovery and Data Mining*, V. S. Tseng, T. B. Ho, Z.-H. Zhou, A. L. P. Chen, and H.-Y. Kao, Eds. Cham: Springer International Publishing, 2014, pp. 40–52.

[72] B. Huynh, C. Trinh, H. Huynh, T.-T. Van, B. Vo, and V. Snasel, "An efficient approach for mining sequential patterns using multiple threads on very large databases," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 242–251, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197618301404

[73] J. Zhang, Y. Wang, and D. Yang, "CCSpan: Mining closed contiguous sequential patterns," *Knowledge-Based Systems*, vol. 89, pp. 1–13, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2015.06.014

[74] C. Sanchez, G. Schneider, W. Ahrendt, E. Bartocci, D. Bianculli, C. Colombo, Y. Falcone, A. Francalanza, S. Krstić, J. M. Lourenço, D. Nickovic, G. J. Pace, J. Rufino, J. Signoles, D. Traytel, and A. Weiss, "A survey of challenges for runtime verification from advanced application domains (beyond software)," *Formal Methods in System Design*, vol. 54, no. 3, pp. 279–335, 2019. [Online]. Available: https://doi.org/10.1007/s10703-019-00337-w

[75] Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem, "Runtime Verification of Component-Based Systems," in *Software Engineering and Formal Methods*, G. Barthe, A. Pardo, and G. Schneider, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 204–220.

[76] M. Štula, D. Stipaničev, and L. Šerić, "Multi-Agent Systems in Distributed Computation," in *Agent and Multi-Agent Systems. Technologies and Applications*, G. Jezic, M. Kusek, N.-T. Nguyen, R. J. Howlett, and L. C. Jain, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 629–637.

[77] A. Legay, A. Lukina, L. M. Traonouez, J. Yang, S. A. Smolka, and R. Grosu, *Statistical Model Checking.* Cham: Springer International Publishing, 2019, pp. 478–504. [Online]. Available: https://doi.org/10.1007/978-3-319-91908-9{_}23

[78] S. M. S. M. Lynch, *Bayesian statistics*, 2020.

[79] P. Singer, D. Helic, A. Hotho, and M. Strohmaier, "Hyptrails: A bayesian approach for comparing hypotheses about human trails on the web," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 1003–1013.

[80] J. K. Grewal, M. Krzywinski, and N. Altman, "Markov models—Markov chains," *Nature Methods*, vol. 16, no. 8, pp. 663–664, 2019. [Online]. Available: https://doi.org/10.1038/s41592-019-0476-x

[81] F. C. De Sousa, N. C. Mendonça, S. Uchitel, and J. Kramer, "Detecting implied scenarios from execution traces," *Proceedings - Working Conference on Reverse Engineering, WCRE*, pp. 50–59, 2007.

[82] F. H. Fard and B. H. Far, "Detection and verification of a new type of emergent behavior in multiagent systems," *INES 2013 - IEEE 17th International Conference on Intelligent Engineering Systems, Proceedings*, pp. 125–130, 2013.

[83] V. Rehak, P. Slovak, J. Strejcek, and L. Hélouët, "Decidable Race Condition and Open Coregions in HMSC," in *GT-VMT - 9th International Workshop on Graph Transformation and Visual Modeling Techniques*, Paphos, Cyprus, Mar. 2010. [Online]. Available: https://hal.inria.fr/inria-00589712

[84] R. Abdallah, L. Hélouët, and C. Jard, "Distributed implementation of message sequence charts," *Software and Systems Modeling*, vol. 14, no. 2, pp. 1029–1048, 2015. [Online]. Available: http://dx.doi.org/10.1007/s10270-013-0357-1

[85] A. Do-Mai, T. Diep, and N. Thoai, "Race condition and deadlock detection for large-scale applications," in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, 2016, pp. 319–326.

[86] A. J. Mooij, N. Goga, and J. M. T. Romijn, "Non-local Choice and Beyond: Intricacies of MSC Choice Nodes," in *Fundamental Approaches to Software Engineering*, M. Cerioli, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 273–288.

[87] A. Slama, Z. S. H. Abad, and B. Far, "Detection of Asynchronous Concatenation Emergent Behaviour in Multi-Agent Systems," in *Agents and Multi-Agent Systems: Technologies and Applications 2021*, G. Jezic, J. Chen-Burger, M. Kusek, R. Sperka, R. J. Howlett, and L. C. Jain, Eds. Singapore: Springer Singapore, 2021, pp. 77–88.

[88] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF Open-Source Data Mining Library Version 2," in *Machine Learning and Knowledge Discovery in Databases*, B. Berendt, B. Bringmann, É. Fromont, G. Garriga, P. Miettinen, N. Tatti, and V. Tresp, Eds. Cham: Springer International Publishing, 2016, pp. 36–40.

[89] F. Hendijani Fard and B. H. Far, *On the Usage of Network Visualization for Multiagent System Verification*. Cham: Springer International Publishing, 2014, pp. 201–228. [Online]. Available: https://doi.org/10.1007/978-3-319-13590-8_10

[90] H. Ben-Abdallah and S. Leue, "MESA: Support for scenario-based design of concurrent systems," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 1384, no. June 2014, pp. 118–135, 1998.

[91] E. Letier, J. Kramer, J. Magee, and S. Uchitel, "Monitoring and control in requirements analysis," *Proceedings - 27th International Conference on Software Engineering, ICSE05*, pp. 382–391, 2005.

Appendix A

Permission of copyright material

SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Jul 21, 2021

This Agreement between Mrs. Anja Slama ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	5113821141295
License date	Jul 21, 2021
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Detection of Asynchronous Concatenation Emergent Behaviour in Multi-Agent Systems
Licensed Content Author	Anja Slama, Zahra Shakeri Hossein Abad, Behrouz Far
Licensed Content Date	Jan 1, 2021
Type of Use	Thesis/Dissertation
Requestor type	non-commercial (non-profit)
Format	electronic
Portion	full article/chapter

Will you be translating?	no
Circulation/distribution	1 - 29
Author of this Springer Nature content	yes
Title	Detection of Emergent Behaviour in Distributed Software Systems using Data Analysis Techniques
Institution name	University of Calgary
Expected presentation date	Jul 2021
Requestor Location	Mrs. Anja Slama [redacted] [redacted] Canada Attn: Mrs. Anja Slama
Total	0.00 USD

Terms and Conditions

Springer Nature Customer Service Centre GmbH
Terms and Conditions

This agreement sets out the terms and conditions of the licence (the **Licence**) between you and **Springer Nature Customer Service Centre GmbH** (the **Licensor**). By clicking 'accept' and completing the transaction for the material (**Licensed Material**), you also confirm your acceptance of these terms and conditions.

1. **Grant of License**

   1. 1. The Licensor grants you a personal, non-exclusive, non-transferable, world-wide licence to reproduce the Licensed Material for the purpose specified in your order only. Licences are granted for the specific use requested in the order and for no other use, subject to the conditions below.

1. 2. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of the Licensed Material. However, you should ensure that the material you are requesting is original to the Licensor and does not carry the copyright of another entity (as credited in the published version).

1. 3. If the credit line on any part of the material you have requested indicates that it was reprinted or adapted with permission from another source, then you should also seek permission from that source to reuse the material.

2. Scope of Licence

2. 1. You may only use the Licensed Content in the manner and to the extent permitted by these Ts&Cs and any applicable laws.

2. 2. A separate licence may be required for any additional use of the Licensed Material, e.g. where a licence has been purchased for print only use, separate permission must be obtained for electronic re-use. Similarly, a licence is only valid in the language selected and does not apply for editions in other languages unless additional translation rights have been granted separately in the licence. Any content owned by third parties are expressly excluded from the licence.

2. 3. Similarly, rights for additional components such as custom editions and derivatives require additional permission and may be subject to an additional fee. Please apply to Journalpermissions@springernature.com/bookpermissions@springernature.com for these rights.

2. 4. Where permission has been granted **free of charge** for material in print, permission may also be granted for any electronic version of that work, provided that the material is incidental to your work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.

2. 5. An alternative scope of licence may apply to signatories of the STM Permissions Guidelines, as amended from time to time.

3. Duration of Licence

3. 1. A licence for is valid from the date of purchase ('Licence Date') at the end of the relevant period in the below table:

Scope of Licence	Duration of Licence
Post on a website	12 months
Presentations	12 months
Books and journals	Lifetime of the edition in the language purchased

4. Acknowledgement

4. 1. The Licensor's permission must be acknowledged next to the Licenced Material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract, and must be hyperlinked to the journal/book's homepage. Our required acknowledgement format is in the Appendix below.

5. Restrictions on use

5. 1. Use of the Licensed Material may be permitted for incidental promotional use and minor editing privileges e.g. minor adaptations of single figures, changes of format, colour and/or style where the adaptation is credited as set out in Appendix 1 below. Any other changes including but not limited to, cropping, adapting, omitting material that affect the meaning, intention or moral rights of the author are strictly prohibited.

5. 2. You must not use any Licensed Material as part of any design or trademark.

5. 3. Licensed Material may be used in Open Access Publications (OAP) before publication by Springer Nature, but any Licensed Material must be removed from OAP sites prior to final publication.

6. Ownership of Rights

6. 1. Licensed Material remains the property of either Licensor or the relevant third party and any rights not explicitly granted herein are expressly reserved.

7. Warranty

IN NO EVENT SHALL LICENSOR BE LIABLE TO YOU OR ANY OTHER PARTY OR ANY OTHER PERSON OR FOR ANY SPECIAL, CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ARISING OUT OF OR IN CONNECTION WITH THE DOWNLOADING, VIEWING OR USE OF THE MATERIALS REGARDLESS OF THE FORM OF ACTION, WHETHER FOR BREACH OF CONTRACT, BREACH OF WARRANTY, TORT, NEGLIGENCE, INFRINGEMENT OR OTHERWISE (INCLUDING, WITHOUT LIMITATION, DAMAGES BASED ON LOSS OF PROFITS, DATA, FILES, USE, BUSINESS OPPORTUNITY OR CLAIMS OF THIRD PARTIES), AND
WHETHER OR NOT THE PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THIS LIMITATION SHALL APPLY NOTWITHSTANDING ANY FAILURE OF ESSENTIAL PURPOSE OF ANY LIMITED REMEDY PROVIDED HEREIN.

8. Limitations

8. 1. ***BOOKS ONLY:*** Where **'reuse in a dissertation/thesis'** has been selected the

following terms apply: Print rights of the final author's accepted manuscript (for clarity, NOT the published version) for up to 100 copies, electronic rights for use only on a personal website or institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/).

8. 2. For content reuse requests that qualify for permission under the STM Permissions Guidelines, which may be updated from time to time, the STM Permissions Guidelines supersede the terms and conditions contained in this licence.

9. Termination and Cancellation

9. 1. Licences will expire after the period shown in Clause 3 (above).

9. 2. Licensee reserves the right to terminate the Licence in the event that payment is not received in full or if there has been a breach of this agreement by you.

Appendix 1 — Acknowledgements:

For Journal Content:
Reprinted by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication)

For **Advance Online Publication papers:**
Reprinted by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].)

For Adaptations/Translations:
Adapted/Translated by permission from [**the Licensor**]: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication)

Note: For any republication from the British Journal of Cancer, the following credit line style applies:

Reprinted/adapted/translated by permission from [**the Licensor**]: on behalf of Cancer Research UK: : [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT]** (year of publication)

For **Advance Online Publication** papers:
Reprinted by permission from The [**the Licensor**]: on behalf of Cancer Research UK: [**Journal Publisher** (e.g. Nature/Springer/Palgrave)] [**JOURNAL NAME**] [**REFERENCE CITATION** (Article name, Author(s) Name), [**COPYRIGHT**] (year of publication), advance online publication, day month year (doi: 10.1038/sj.

[JOURNAL ACRONYM])

For Book content:
Reprinted/adapted by permission from [**the Licensor**]: [**Book Publisher** (e.g. Palgrave Macmillan, Springer etc) [**Book Title**] by [**Book author**(s)] [**COPYRIGHT**] (year of publication)

Other Conditions:

Version 1.3

Questions? customercare@copyright.com or +1-855-239-3415 (toll free in the US) or +1-978-646-2777.