Kiel University
Faculty of Computer Science

A Measurement Framework for the Assessment of Self-Adaptation and Self-Organisation in Systems with Decentralised Configurations

Dissertation

submitted to obtain the degree of

DOCTOR OF ENGINEERING (DR.-ING.)

Author:
Jürgen-Martin Goller

Supervisor:
Prof. Dr.-Ing. Sven Tomforde

Winsen/Luhe 2023

1. Reviewer: Prof. Dr.Ing. Sven Tomforde
2. Reviewer: Prof. Dr. Christian Krupitzer

Day of the oral examination: 15.02.2024

Zusammenfassung

Schlagwörter Framework, Selbstadaption, Selbstorganisation, Metriken, Konfigurationen, Fehler-Ursache-Analyse, Systemverhalten, Verkehrsanalyse.

In vielen Bereichen unseres Lebens treffen wir auf technische Geräte, die immer mehr in der Lage sind, selbstständig Entscheidungen zu treffen und ohne das Eingreifen eines Menschen ihr Verhalten an sich ändernde Umweltbedingungen anzupassen. Ein besonders prominentes Beispiel dafür sind selbstfahrende Autos, die mitlerweile jeder große Fahrzeughersteller im Angebot hat.

Der nächste Schritt der Entwicklung autonomer Systeme ist ihr Zusammenwirken, also die Vernetzung von mehreren autonomen Geräten, die gemeinsam gegebene Aufgaben bewältigen. Ein Beispiel sind hier intelligente, vernetzte Kameras, die über ein großes Areal Objekte verfolgen können.

Wenn ein System Entscheidungen selbsttätig trifft, dann wird es um so wichtiger, dass ein menschlicher Benutzer das Verhalten des Systems versteht. Denn Unverständnis führt sehr häufig zu Ablehnung. Insbesondere betrifft dies Situationen, in denen eine Änderung des Systemverhaltens unerwartet auftritt.

Ausgehend von der Hypothese, dass unerwartete Änderungen im Verhalten ein Resultat von ungewöhnlichem Adaptionsverhalten sind, stellt sich die Frage, wie sich ungewöhnliches Adaptionsverhalten erkennen lässt. Als Antwort darauf stellt diese Arbeit ein Framework aus verschiedenen Metriken vor, welche Änderungen in den Konfigurationen des Systems in eine Bewertung des Adaptionsverhaltens überführen und dieses messbar machen. Dafür werden die Metriken des Frameworks formal definiert, detailliert erklärt und mit existierenden Metriken verglichen. Im Anschluss werden verschiedene Szenarien aus unterschiedlichen Domänen vorgestellt und ausgewertet. Ein besonderes Augenmerk liegt hier auf spezielle Ereignisse, die zu ungewöhnlichen Selbstadaptionsverhalten führen. Die Ergebnisse werden dann ausgewertet und eingeordnet. Abschließend wird eine Erweiterung des Frameworks dargestellt, mit in Fehler-Ursache-Analysen Hilfestellung geben kann. Das Vorgehen dafür wird dann verschiedenen Szenarien veranschaulicht und dabei gezeigt, dass das Framework dafür geeignet ist.

Abstract

Keywords Framework, self-adaptation, self-organisation, metrics, configurations, root cause analysis, system behaviour, traffic analysis.

In many areas of our lives, we find technical devices that are able to autonomously make decisions and adapt their behaviour to changing conditions without the intervention of a human being. A particularly prominent example is self-driving cars, which every major car manufacturer offers today. The ability to drive fully autonomously in any traffic situation is being worked on intensely.

The next step in the evolution is the collaboration of multiple autonomous devices that collectively manage a given task. Examples include smart cameras that can trace objects in large areas.

When a system makes decisions autonomously, then it becomes even more important that the user understands the system's behaviour. A lack of understanding often leads to rejection. This particularly applies to situations where the system behaviour changes unexpectedly.

Starting from the hypothesis that unexpected changes in behaviour result from unusual self-adaptations, the question arises as to how unusual self-adaptations can be detected. As an answer to this question, this thesis presents a framework of different metrics that translates change in configuration parameters in a system to an assessment and quantification of the self-adaptation behaviour. The metrics contained in the framework are formally defined, explained in detail and compared to existing metrics.

Then, several evaluation scenarios from different domains are presented and evaluated with the framework. Here, a focus is set on specific events in the scenarios that lead to unusual self-adaptation behaviour. The results are then evaluated and brought into context.

The thesis then closes with a presentation of an extension to the framework that can aid in root cause analysis. The procedure for such an analysis is then illustrated in different scenarios, showing that the framework is eligible for this application.

List of Abbreviations

AC	Autonomic Computing
AI	Artificial intelligence
CM	Control mechanism
CPU	Central processing unit
DPSS	Decentralised Progressive Signal System
HBOS	Histogram-based Outlier Score
IPv4	Internet protocol version 4
IPv6	Internet protocol version 6
IT	Information technology
MAPE	Monitoring, analysing, planning and executing
MAPE-K	Monitoring, analysing, planning, executing and knowledge
MAS	Multi-agent system
OC	Organic Computing
OTC	Organic Traffic Control
PS	Productive system
RCA	Root cause analysis
SA	Self-adaptation
SASO	Self-adaptation and self-organisation
SO	Self-organisation
SOS	System of systems
SuOC	System under Observation & Control

Table 1: List of abbreviations

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation - *"The Rise of Complexity"*

In 1965, Gordon Moore observed that integrated circuits were becoming increasingly complex and established the underlying mathematical rule, which later would be called Moore's law: the number of transistors in a dense integrated circuit doubles about every two years [145]. While this law is slowly reaching physical limits for integrated circuits [155], its quintessence still holds: technical systems tend to become more and more complex [25]. One example of this phenomenon is the internet. Counting the assigned IPv4 addresses, there were about 72 million devices connected to the internet in 2000. Twenty years later, all of the 4.294.967.296 possible addresses are considered allocated [265]. But not only the number of participating devices grows continuously. Since 2000, the number of published RFC documents – which describe methods, behaviours, research, or innovations applicable to the working of the internet and internet-connected systems – is at an average of about 280 per year. And while the first electronic engine control unit for cars was introduced in the 1950s [258], contemporary cars can have more than 100 electronic control devices, are connected to the internet, and run with millions of lines of software code [267, 150].

Another observation on complexity is given by *Glass*, who states that IT complexity is indirectly related to functionality in that a 25% increase in functionality increases complexity by 100% [70].

The ever-growing complexity can reach dimensions where classical approaches to system designs are highly problematic or even impossible to apply. This regards the modelling of such systems [170], their implementation [91] or their maintenance [110]. And not only technical systems are becoming more complex, but also their applications. Traffic management systems [144], autonomous driving [173], and the Internet of Things [253]

are examples of this progression. Also, the increment of complexity on the application side reveals the limits of classical systems. There are applications for which monolithic system designs are inapplicable. *Rao & Georgeff* show this with an air traffic management system [182].

Based upon ideas from cybernetics [104], in the early 2000s, different research initiatives such as *Proactive Computing* [220], *Autonomic Computing* [105], *Complex Adaptive Systems* [119], *Pervasive/Ubiquitous Computing* [156, 238] or *Organic Computing* [229] presented concepts to cope with these problems. Big monolithic systems were replaced with sets of smaller, interacting, and more and more autonomous systems that work together to reach the system goal. Such systems are able to autonomously make decisions at runtime that previously were made by the developer at design time. These decisions are typically not made by a central instance, but the decision process is distributed to the subsystems. The subsystems have knowledge about their local environment and their own state. All mentioned research initiatives propose methods that allow these systems to adapt their behaviour and to organise with each other. In this thesis, we will refer to such systems as self-adaptive and self-organising (SASO).

Due to these capabilities, SASO systems can react to unforeseen events such as changes in the environment, hardware defects, communication problems, internal problems, and others. While classical system designs will usually face a heavy decrease in the system performance or might even fail completely, a SASO system will make decisions about its behaviour and structure with the goal of maintaining an acceptable performance under such circumstances. We will see some examples of such systems in Chapter 2.3.1 and 2.3.2.

1.2 The Problem - *"Should it do this?"*

When humans use a technical system, they have specific expectations about what should happen. The main expectation is that the system reaches its designated system goal and creates the desired outcome. When using a refrigerator, the expectation is that the groceries are cooled, and a navigation device is expected to find an acceptable route to the destination. Another expectation is replicability: the refrigerator should maintain its cooling capability and keep the set temperature, and the navigation device should deliver the same route when asked repeatedly under the same traffic conditions. Systems that do not conform to the expectations are perceived as unreliable or even defective and might no longer be accepted by the user. Thus, user acceptance is an essential property of technical systems interacting with humans. Scientific insight into these aspects is given by the research field of *human computer interaction* [50].

An important part of user acceptance is the user experience [95]. This deals with the user's perception of the interactions with the system and questions like: *"How easy is the system to use?"*, *"Is it convenient?"* or *"Do I understand what the system does and why it does that?"*. Especially the last question becomes significant when interacting with systems that make decisions [207]. So, humans interacting with decision-making systems on a regular basis will create expectations of what these decisions look like based on their experience. And if these expectations are suddenly no longer met, user acceptance will decrease. The best way to maintain the level of user acceptance is to give explanations of what is happening [19]. A simple example will illustrate that: Imagine driving to work in your car through a city with a smart traffic control system. You see a traffic light in the distance becoming red, and you stop your car. From your daily commute on this route, you know that this specific light will stay red for about 20 seconds. But today, it is already over a minute. You wonder if the traffic light is broken. You think about continuing your way even if it is still red. Nobody is watching. Now, suppose the traffic control system informs you that there is a major accident three blocks ahead, and your traffic light is red to allow the ambulance to go through. In that case, you might reconsider your possible actions, and you might understand that the smart traffic control is, in fact, working as wanted.

In this example, the user only observes the system and still has control of his actions. He still can decide to continue his drive. In scenarios where the user does not have this freedom, the acceptance becomes crucial. An example of this case are autonomous ferries [5]. Once the user is on board, he is at the mercy of the system. The only choice for the user is between using the ferry or not. In this case, unexplained deviations from the expected behaviour can lead to decreasing passenger numbers and, eventually, commercial problems for this ferry business.

We see that changes in the observed behaviour of a technical system should be explained. But to give a self-explanation, the system needs to be aware of *when* it should do that. In SASO systems, two major factors can lead to a change in the observable behaviour: emergence and self-adaptation/self-organisation [19], where, in some cases, emergence will lead to self-adaptation [96]. Therefore, one indicator for an event that should trigger the self-explanation process is the detection of a noteworthy self-adaptation event.

Another reason to have indicators for unusual self-adaptation events is the choice of adaptation strategy within a SASO system. The question here is to decide whether the current adaptation strategy is still appropriate or at what point the strategy should be changed. An increase in noteworthy self-adaptation events in the system can be a pointer towards the necessity for a new strategy.

This leads us to the problem that this thesis tries to approach:

How can noteworthy self-adaptation events be detected on a system-wide level at runtime and without deep system knowledge?

During normal operation, a SASO system will either undergo no self-adaptation at all or will show some basic noise of self-adaptation. Therefore, a *noteworthy* self-adaptation event is one that is clearly distinguishable from the observed self-adaptation efforts during normal operation. Self-adaptation events that are limited to a single subsystem may or may not affect the behaviour of the whole system. Since we are interested in the observable behaviour of the whole system, we want to identify those events that eventually are not limited to single subsystems. That is what *system-wide level* means. Such adaptations should be detected without any substantial delay. This is the only way the user can decide whether an intervention is necessary. This is what *detection at runtime* means. Furthermore, an approach to detect the events that uses *no deep knowledge* about the system can be easily transferred to other system without the need for major modification.

1.3 Scientific Focus and Contribution

The scientific focus of this thesis and its contribution to scientific knowledge is as follows:

- **Metrics:** This thesis presents several new metrics that are based on the analysis of configuration changes of SASO systems at runtime. The metrics aim to give insight into the self-adaptation process of the system and to identify unusual events on a system-wide scale. They can be applied to various system types and do not require special domain knowledge. In detail, they meet the following requirements:
  - **Req1:** They do not require expert knowledge. They can be implemented without knowledge about the system goal, the application domain and other aspects of expert knowledge.
  - **Req2:** They do not require knowledge about internal system states.
  - **Req3:** They do not require knowledge about the structure of the system or the dependencies of its components.
  - **Req4:** They do not require knowledge about the outcome of self-adaptation processes.
  - **Req5:** They can be evaluated at runtime. The metrics identify unusual events with no or an acceptable delay.
- **Measurement Framework:** The thesis analyses the behaviour of the metrics in several SASO systems and shows that metrics for themselves seldom detect such

events with sufficient precision in every situation. To cope with this, the thesis illustrates how a measurement framework containing several metrics increases the detection rate.

- **Root Cause Analysis:** Finally, the thesis shows how the measurement framework can aid a root cause detection system in finding sources of problems in a SASO system. This is done by combining the calculation of the metrics with a scoring system that assigns each subsystem a root cause score. Such root causes can then be presented to the user as a basis for future self-explaining SASO systems.

1.4 Contained Publications

This thesis is based on the following publications by the author:

1. Sven Tomforde and Martin Goller: **To Adapt or Not to Adapt: A Quantification Technique for Measuring an Expected Degree of Self-Adaptation** – In MDPI Computers, Special Issue on Applications in Self-Aware Computing Systems and Their Evaluation 2020 [222].
   This position paper shows the necessity for a measurement framework, and describes the challenges of defining such a framework.
2. Martin Goller and Sven Tomforde: **Towards a Continuous Assessment of Stability in (Self-)Adaptation Behaviour** – 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) [74].
   In this paper, the *Configuration Stability* metric (Chapter 4.3.4) is defined and evaluated in the Flocking scenario (Chapter 5.1.1) and the Artificial Road Network - Road Block scenario (Chapter 5.1.3). This contributes to the task of defining metrics that fulfil **Req 1** to **Req 5**.
3. Martin Goller and Sven Tomforde: **On the Stability of (Self-)Adaptive Behaviour in Continuously Changing Environments: A Quantification Approach** – In Elsevier Array 2021 [77].
   This paper provides a deeper evaluation of the *Configuration Stability* and introduces the Artificial Road Network - Rush Hour scenario (Chapter 5.1.4).
4. Martin Goller and Sven Tomforde: **Assessment of Configuration Stability and Variability in Collections of Self-Adaptive Systems** – 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) [75].
   Here, the Configuration Variability (Chapter 4.3.5) is defined and evaluated in the Flocking scenario and the Artifical Road Network - Rush Hour scenario. This, again, contributes to the task of defining metrics that fulfil **Req 1** to **Req 5**.
5. Martin Goller and Sven Tomforde: **Beyond Homeostasis: A Novel Approach**

for Assessing the Stability and Coherence of Self-Adaptive Systems – 2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC / PiCom / CBDCom / CyberSciTech) [76].

This paper establishes the Coherence Metrics (Chapter 4.3.6) and evaluates them in the Flocking scenario, the Artificial Road Network - Rush Hour scenario, and the Game of Life scenario (Chapter 5.1.7). This is another contribution to the task of defining metrics that fulfil **Req 1** to **Req 5**.

6. Martin Goller and Sven Tomforde: **Identifying Adaptation Changes in Collections of Self-Adaptive Systems** – 2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) [78].
   In this paper, we present the extension of the framework for the application in a root cause analysis (Chapter 6). Here, we see how the framework contributes to our central problem statement: Identifying noteworthy self-adaptation events on a system-wide level at runtime in different systems.
7. Martin Goller and Sven Tomforde: **Runtime Assessment of the Parameter Utilisation in Adaptive Systems** – 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) [79].
   The *Global Parameter Usage* and *Average Parameter Usage* metrics (Chapter 4.3.7) are introduced in this paper together with an evaluation in the Artificial Road Network - Rush Hour scenario and the Smart Camera scenario (Chapter 5.1.2). This is one more contribution to the task of defining metrics that fulfil **Req 1** to **Req 5**.
8. Martin Goller, Ingo Thomsen, Ghassan Al-Falouji, and Sven Tomforde: **Abnormal Behaviour Detection of Self-Adaptive Agents in Traffic Environments** – 2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C) [73].
   This paper presents the evaluation of the *Configuration Stability*, the *Configuration Coherence*, and the *Global Parameter Usage* in the Life-Like Road Network scenario (Chapter 5.1.5) and the Maritime Traffic scenario (Chapter 5.1.6). Again, this contributes to our central problem statement: Identifying noteworthy self-adaptation events on a system-wide level at runtime in different systems. In this case with a focus on real-world applications.

1.5 Organisation of the Thesis

The core parts of the train of thought in this thesis are shown in Figure 1.1: the description of the system model that we use, the measurement framework that uses the output from the system, and the evaluation and application of the framework.

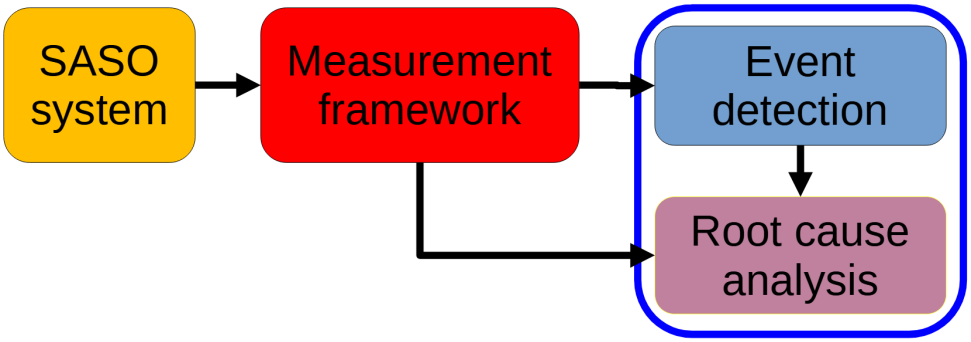


Figure 1.1: The main parts of the thesis: a **measurement framework** (Chapter 4) collects data from a **SASO system** (Chapter 2), creates several time series as output, which are then **evaluated** with a focus on event detection (Chapter 5) and finally **applied** in a root cause analysis (Chapter 6).

In detail, the thesis is structured as follows: In Chapter 2, we will give the overall context that this thesis lies within. In Chapter 2.1, we will learn about complexity, followed by self-adaptation and self-organisation in Chapter 2.2, an overview of related research fields (Chapter 2.3), and our system model in Chapter 2.4. Chapter 3 gives an overview of existing metrics for self-adaptation and self-organisation and other assessment approaches for self-adaptive systems. Then, we come to the theoretic core in Chapter 4, where we will introduce our metrics. With the formulae at hand, we will evaluate the metrics in several scenarios in Chapter 5. Chapter 6 shows the application of the framework in a root cause detection scenario. For this, we will have a short introduction to the domain of root cause detection in Chapter 6.1, followed by the actual application in Chapter 6.3 and a comparison with other approaches (Chapter 6.5). Chapter 7 concludes the thesis with a summary (Chapter 7.1), an outlook on future work and open questions (Chapter 7.2).

Chapter 2

Background

At its core, this thesis deals with complex technical systems. Therefore, we will investigate what complexity is in Chapter 2.1 first. Then, to set the stage for the SASO assessment (i.e. the assessment of the self-adaptation behaviour of the SASO system), we need to talk about the basic context we are in and the underlying notions. For this, Chapter 2.2 will give a short introduction to self-adaptation and self-organisation. After that, we will take a look at two research initiatives that have a major impact on SASO system designs: *Autonomic computing* (Chapter 2.3.1) and *Organic Computing* (Chapter 2.3.2). Next, we will glance at the field of *multi-agent systems* in Chapter 2.3.3 and a few other research fields (Chapter 2.3.4). All this leads to the system model we will use in this thesis. We will discuss it in Chapter 2.4.

2.1 Complexity

What is *complexity*, and what is *complex*? The Merriam-Webster online dictionary defines the adjective *complex* as "hard to separate, analyse, or solve" [266] and as the opposite of *simple*. A synonym would be *complicated*. Researchers either stick with this definition or create one that suits their purposes. Therefore, there is no unique definition of *complexity* among scientists [101].

2.1.1 Complex Systems

SASO systems are built to handle complexity, either the complexity of its environment or the complexity of the problem to solve. If it is the environment, then we could see that environment as a complex adaptive system [63]. Such systems are characterised by several properties [36], such as non-linear interactions, but a strict definition for the term *complex* is not given. A (not very strict) definition based on the interactions is given by

H. Simon, who defines a complex system as a large set of elements that interact in a non-trivial way [209]. Here, the emphasis is on the non-triviality. Other definition approaches to complex systems are based on the presence of emergence in the system. That means the system has properties that are only observable at the macro level. *Cotsaftis* argues that it is the emergence that distinguishes complex systems from complicated ones [37]. *Standish* states that complexity in complex systems can have two connotations: quality and quantity. If seen as quality, the complexity of a system refers to the presence of emergence. If seen as quantity, complexity refers to the amount of information needed to specify the system [215].

For this thesis, the complexity of a system is of minor importance. Therefore, we accept any definition of *complex systems* that defines distributed control systems as complex.

2.1.2 Complex Problems

The actual control problem that the SASO system should solve is given by its goal or utility function. From this point of view, the complexity of the problem refers to its computational complexity [203], where we have strict definitions at hand. Examples here are the *Kolmogorov complexity* of strings or the *time complexity* of algorithms. Basically, the Kolmogorov complexity of a string is the length of the shortest algorithm that produces that string [112]. The time complexity of an algorithm, on the other hand, describes the relation between the size of the input of the algorithm and the time it needs to run [211].

If we step back and take a look from a distance, we see that the utility function is given by the system designer. The designer interprets the real-world problem and translates it into a computational problem. At this point, we have to note that the term complexity in "real-world problems" is usually not the same as computational complexity. A mathematical definition of problem complexity at this abstraction level is given by *Saladoa & Nilchiania* [198], who define the complexity of a problem as a function of the size of the solution space, which takes the requirements of the problem and their conflicts into account.

Another point of view on problem complexity is given by agile project management methods. Here, a problem is defined as complex if there are sufficient uncertainties about the requirements or the possible approaches to solve the problem. Problems that can be solved with upfront planning and expert knowledge are considered *complicated* [140]. For complex problems, agile management methods (e.g. Scrum [269]) propose project teams that are self-organising and self-adaptive – a SASO system consisting of humans.

A general overview of different notions and the etymology of the words *complex* and *complexity* can be found in [6].

2.2 Self-Adaptation & Self-Organisation

Self-adaptation and self-organisation are processes within systems that are established and managed by the system itself. Such processes and properties that are caused and maintained by a system itself are called self-* or self-x properties [270]. In the scientific literature, several self-* properties are investigated. The CAS group wiki lists more than 30 properties in six categories: healing, reconfiguring, optimising, autonomy, sense of self, and protecting [270]. Several authors use different ideas to classify and categorise these properties. While *Hermann et al.* subsume self-adaptation and self-organisation under the notion of self-management [89], the autonomic computing (AC) initiative (see Chapter 2.3.1) defines self-configuration, self-optimisation, self-healing and self-protection as the parts of self-management [105]. On the other hand, *Salehie & Tahvildari* introduce a hierarchy of self-* properties where the four AC properties are aspects of the general self-adaptation property and are based on self-awareness [199].

Since the focus of the thesis is on self-adaptation and self-organisation, we will now investigate these two notions more closely.

2.2.1 Self-Adaptation

For the introduction to self-adaptation, we will adhere to the book *An Introduction to Self-Adaptive Systems* by *Weyns* [241]. Unless stated otherwise, all statements in this chapter are taken from there.

The field of software engineering has seen several fundamental shifts of thinking and focus. The first is the shift in focus from development time to runtime. Before that, the focus was on getting everything right during the development. If a running system encountered problems at runtime, it was taken offline, fixed and then redeployed. But today, most public-facing systems must be continuously available. Downtimes for maintenance are unacceptable. This requires the system to be modifiable at runtime.

The second shift is the increasing level of uncertainties that modern systems face. In the past, software was developed for a known environment, and most components were under the control of the developers. Today, systems operate in much more uncertain contexts: unpredicted changes in load, unstable communication networks, faults that arise from interaction with other systems or external attacks.

The third shift is the interest in automation to reduce costs. Although the costs for acquiring and developing increasingly complex systems are dropping, the total cost of ownership has been rising in the past. The reason for this was the need for system administration which took up larger and larger fractions of the operational budget. The automation of routine tasks performed by the administrators countervails that. Moreover, for today's complex systems, complete human oversight and control is simply not possible.

Finally, the fourth shift is the commoditisation of artificial intelligence (AI). While mostly limited to special niches (e.g. robotics) in the past, the increasing availability of planners, machine learning, genetic algorithms, and game-theoretic decision systems has made it possible to harness sophisticated reasoning and learning mechanisms in support of automation.

These shifts lead to several crucial questions, such as: *"What are the unifying principles of self-adaptive systems", "How to ensure that system requirements are met even if they change", "How to prevent the adaptation from getting out of hand"*, or *"How to create trust in systems without human oversight?"* The discipline of self-adaptive systems attempts to answer these questions. For this, ideas from a wide range of fields are exerted, from control theory over biology and ecology to software architecture and AI, and many more.

Due to this multiplicity of influences, there is no general agreement on a definition of the notion of *self-adaptation*. However, *Weyns* presents two basic principles that determine what a self-adaptive system is:

Principle 2.2.1 (External principle)**.** *A self-adaptive system is a system that can handle changes and uncertainties in its environment, the system itself, and its goals autonomously (i.e. without or with minimal required human intervention).*

Principle 2.2.2 (Internal principle)**.** *A self-adaptive system comprises two distinct parts: the first part interacts with the environment and is responsible for the domain concerns — i.e. the concerns of users for which the system is built; the second part consists of a feedback loop that interacts with the first part (and monitors its environment) and is responsible for the adaptation concerns – i.e. concerns about the domain concerns.*

Based on these two principles, *Weyns* gives a conceptual model of a self-adaptive system, which we see in Figure 2.1. His model comprises four essential elements: environment, managed system, feedback loop and adaptation goals.

The **environment** refers to the part of the external world with which a self-adaptive system interacts and in which the effects of the system will be observed and evaluated. The environment can include users as well as physical and virtual elements. The environment is a source of uncertainty. Neither the observations by the sensors nor the effects of the interactions can be guaranteed.

The **managed system** provides the function to the users. For this, it senses and affects the environment using sensors and actuators.

The **adaptation goals** relate to quality properties of the managed system. In general, four principal types of high-level adaptation goals can be distinguished: self-configuration (i.e. systems that can change their configuration automatically during runtime without external intervention), self-optimisation (systems that continually seek ways to improve

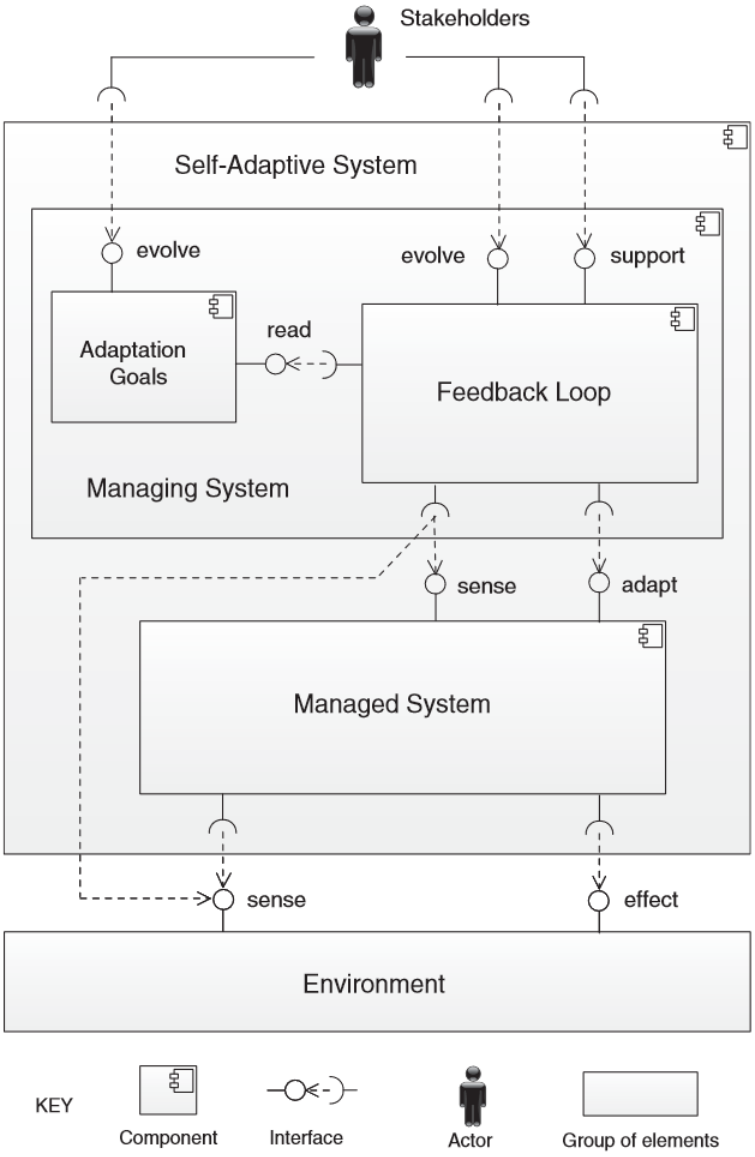


Figure 2.1: The conceptual model of a self-adaptive system by *Weyns* [241]

their performance or reduce their cost), self-healing (systems that detect, diagnose, and repair problems resulting from bugs or failures), and self-protection (systems that defend themselves from malicious attacks or cascading failures).

Finally, the **feedback loop** comprises the adaptation logic that deals with one or more adaptation goals. To realise the adaptation goals the feedback loop monitors the environment and the managed system and adapts the latter when necessary to realise the

adaptation goals. *Weyns* divides this feedback loop into four steps: monitoring, analysing, planning and execution (MAPE) [134] (see Figure 2.2). The research field of *Autonomic Computing* (see Chapter 2.3.1) extends this layout of a feedback loop with a shared knowledge base (see Figure 2.5).

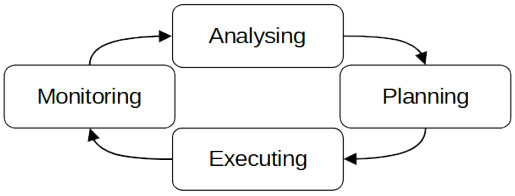


Figure 2.2: The MAPE loop

For a more profound introduction to self-adaptive systems, we recommend *Weyns*'s book as a whole. For a survey on academic research on this topic, we refer to *Macías-Escrivá et al.* [134], who give an overview of approaches, research challenges and applications. For further reading on the more specific topic of developing self-adaptive systems, we refer to *Krupitzer et al.* [115], where the current development approaches are classified and compared. Another noteworthy point is the meta-level: the self-adaptation of the self-adaptations mechanisms. *Krupitzer et al.* give an overview in [116].

2.2.2 Taxonomy of Self-Adaptation

Self-adaptation can be investigated from several points of view, and specific aspects lead to distinct questions. The architectural point of view might ask which components are involved in the self-adaptation, while the reasoning point of view would like to know why the self-adaptation is happening.

To get an overview of these stances, a taxonomy of self-adaptation and its aspects is helpful. The most comprehensive taxonomies are those presented by *Rohr et al.* [190] and *Krupitzer et al.* [117]. Both are based on the 5W + 1H questions by *Salehie & Tahvildari* [199]:

- When to adapt?
- Why do we have to adapt?
- Where do we have to implement change?
- What kind of change is needed?
- Who has to perform the adaptation?
- How is the adaptation performed?

Figures 2.3 and 2.4 show the two proposed taxonomies, which both consist of five dimensions. Although the version of *Krupitzer et al.* is more refined, both approaches use the same aspects for their major dimensions. Let us examine these dimensions closer:

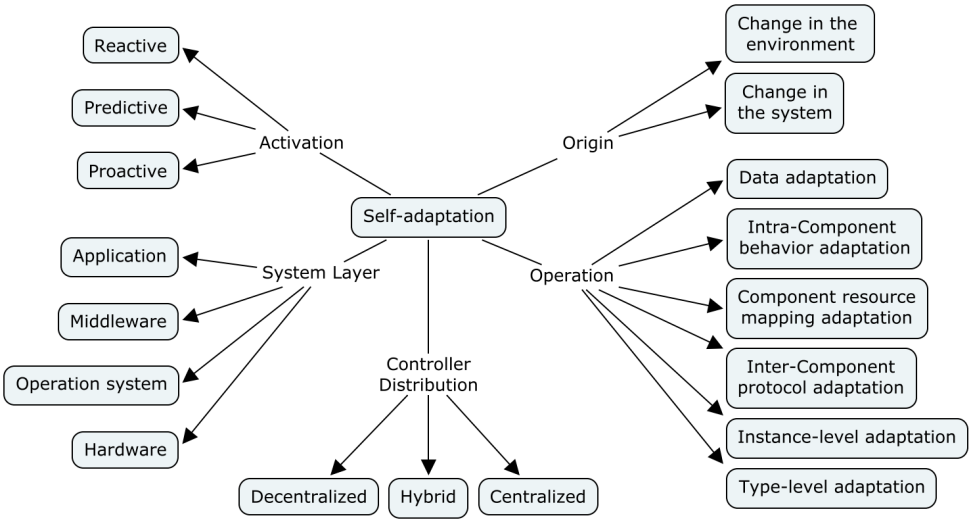


Figure 2.3: The taxonomy of self-adaptation by *Rohr et al.*

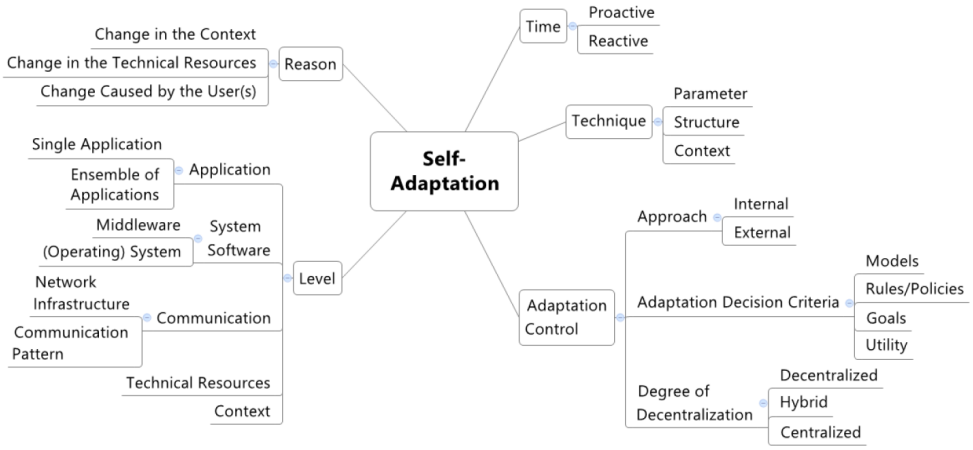


Figure 2.4: The taxonomy of self-adaptation by *Krupitzer et al.*

Activation/Time

The dimension Activation (*Rohr*)/Time (*Krupitzer*) answers the question of *when to adapt*. In this case, *reactive* stands for adaptations that take place after an event is detected and the need for a reaction to this event is identified. In contrast, *predictive* self-adaptation takes place because the system predicts that such an event might occur in the near future. Lastly, *proactive* self-adaptation results from efforts to increase the performance of the system. This takes place without any disruptive event.

Origin/Reason

With Origin (*Rohr*)/Reason (*Krupitzer*), the question of *why we have to adapt* is investigated. Both taxonomies use slightly different ideas to answer the questions. However, all of them agree on the fact that the reason for self-adaption is some kind of change: a change in the environment, the system itself, the context, the resources, or the wishes of the user.

System Layer/Level

The dimension System Layer (*Rohr*)/Level (*Krupitzer*) describes where the self-adaptation is implemented. Both taxonomies distinguish the location of adaptation according to the levels of system architectures, from hardware at the lowest level and the actual application at the top. This dimension answers the questions of *where* and *who*.

Operation/Technique

At the dimension of Operation (*Rohr*)/Technique (*Krupitzer*), the kind of change needed is addressed. Possible alternatives include the adaptation of parameters, the system's structure, the context, the communication between components, or the behaviour of components.

Controller Distribution/Adaptation Control

While the other four dimensions in both taxonomies answer the respective question at the same level, here we see a substantial difference. The *Controller Distribution* dimension in *Rohr's* taxonomy only answers how the adaptation is controlled (see Chapter 2.3.2 for details on this point). *Krupitzer* moves this distinction to a subdimension under the *Adaptation Control* and accompanies it with the aspects of *Approach* and *Decision Criteria*. This fifth dimension answers the question of *how the adaptation is performed*.

2.2.3 Self-Organisation

Self-organisation is observable everywhere around us. The complex mechanisms in a human ovule can lead to the reproduction of this cell. With some time, these new cells will organise themselves to form a human body [216]. Eventually, those human bodies are capable of organising themselves in social structures [60]. Self-organisation exists on all scales: from the microscopic world – water molecules that form snowflakes [157] – to the largest scale where self-organisation is visible in galaxies [33]. Another example is the flocking behaviour of birds. We will take a closer look at that in Chapter 5.1.1.

In scientific literature, the notion of *self-organisation* was introduced by *Ashby* in the 1940s [8]. He used the term to describe the formation of patterns that emerge from the cooperation of individual entities. Since self-organisation is a research subject of many different scientific fields, including biology [46], sociology [60], and computer science [49], there is no exact definition of it that is universally accepted [55]. In this thesis, our focus is on technical systems. One definition for self-organisation in this field is given by *Di Marzo Serugendo et al.*, who define self-organisation as the mechanism or the process enabling a system to change its organisation without explicit external command during its execution time [48]. Another one is presented by *Heylighen*, who defines self-organisation as the spontaneous creation of a globally coherent pattern out of the local interactions between initially independent components [90]. A review of definitions and concepts of self-organisation is given by *Halley & Winkler* [85].

For this thesis, we will use the definition given by *Elmenreich & Meer* [54] for networked systems:

Definition 2.2.3. A self-organising system is a set of entities that achieves a global system behaviour via local interactions between its entities without centralised control.

Together with this definition, *Elmenreich & Meer* distinguish three different approaches to model a self-organising system: differential equations, cellular automata and agent-based systems. For the latter, we refer to Chapter 2.3.3, where we will investigate multi-agent systems.

Differential Equations

A basic example of the self-organisation effects in models described by differential equations [58] are *predator-prey systems* [236]. Such systems contain two types of animals, typically named *rabbits* (the prey) and *foxes* (the predators). The populations of the animals follow four simple rules:

- Rabbits reproduce at a given birth rate ε_1.
- A rabbit dies if caught by a fox. This death rate is γ_1. The number of deaths is proportional to the number of foxes.
- The reproduction rate of the foxes is γ_2. The number of new foxes is proportional to the number of rabbits.
- Foxes die with a given death rate ε_2.

These rules can be compiled into two differential equations:

$$\frac{dN_1}{dt} = N_1(\varepsilon_1 - \gamma_1 N_2) \tag{2.1}$$

$$\frac{dN_2}{dt} = N_2(\varepsilon_2 - \gamma_2 N_1) \tag{2.2}$$

where N_1 and N_2 are the number of rabbits or foxes, respectively. On evaluation, this system shows periodic oscillations of N_1 and N_2. Examples of models of population dynamics that use this approach include predator-prey systems with diseases [129], systems with three species [59], and systems with a differentiation of young and adult predators [127].

Another important example of self-organising systems modelled with differential equations are reaction-diffusion systems. In chemistry, this refers to systems where chemical reactions occur locally, and the resulting substances then spread across a surface [67]. In biological systems, many processes can be described by such systems. A review of pattern formation based on reaction-diffusion equations can be found in [39].

Cellular Automata

Cellular automata [47] are a model for systems where the entities can be represented by cells of a discrete grid, and every cell/entity can only assume a finite number of possible states. Furthermore, the next state of a cell only depends on its current state and the current state of its neighbours.

Applications for cellular automata are the modelling of predator-prey systems [32], traffic jams [152], thermodynamic systems [246] and others.

One of the most prominent examples of cellular automata is *Conway's* Game Of Life [61]. Here, the cells can have two states: dead and alive. The next state of a cell depends on the number of living cells in its neighbourhood. *Conway's* classic rules are:

- The neighbourhood of a cell are the eight adjacent cells.
- A dead cell becomes alive if there are precisely three living cells in the neighbourhood.
- A living cell dies if there are less than two living neighbours.
- A living cell with two or three living neighbours stays alive.
- A living cell with more than three living neighbours dies.

Depending on the starting states of the cells, interesting patterns can emerge: static clusters of living cells, oscillations of death and birth and oscillating patterns that move. With the correct starting states, these yet so simple rules can even form a Turing machine [186]. The aspect of self-organisation in cellular automata lies in the fact that random starting states can produce specific patterns.

2.2.4 Properties of Self-Adaptive & Self-Organising Systems

As a short summary, with respect to the aforementioned notions of self-adaptation and self-organisation, and according to *Banzhaf* [10] and *Nakagawa* [153], we can state that SASO systems have the following properties:

- They consist of several distributed system parts.
- They show a strong correlation of the system parts.
- There are positive and negative feedback loops.
- They are resilient against errors and disturbances.
- They exhibit emergence.
- They evaluate their own behaviour, goals, and internal states.
- And they can dynamically change their behaviour.

2.3 Research Fields

To bring the general concepts of self-adaptation and self-organisation to actual technical systems, several research initiatives have created theoretical foundations and investigated ways of applications. The two initiatives with the most influence are *Autonomic Computing*, which we will introduce in Chapter 2.3.1, and the *Organic Computing* initiative (Chapter 2.3.2).

In Chapter 2.3.4, we will see other research fields with contributions to this topic. This thesis deals with decentralised configurations. Hence, we will investigate multi-agent systems in Chapter 2.3.3 as prototypes for such decentralisation.

2.3.1 Autonomic Computing

In 2001, IBM published an article [94] stating that the growing software complexity would become a significant obstacle to further progress. Millions of lines of code, large heterogeneous environments and more and more interconnected systems were reaching a point at which even the most skilled humans could not keep pace with tasks necessary to keep everything running. Focused on the management and automation of data centres and based on the notion of the autonomic nervous system, which controls low-level functions in the human body, *Kephart & Chess*, researchers at IBM, introduced the concept of *autonomic computing* (AC) [105] in 2003. The core of the concept is the self-management of autonomic computing systems which moves maintenance tasks and other details of operation from the administrator to the system itself. *Kephart & Chess* specify four aspects necessary for self-management: *self-configuration, self-optimisation, self-healing* and *self-protection*.

In the context of AC, **self-configuration** means that only high-level configuration policies are given by an administrator. The configuration of systems and components is automated and follows those policies.

For *Kephart & Chess*, **self-optimisation** is the ability to identify and seize opportunities to improve performance or reduce cost. To achieve this, AC systems can change their own parameters, outsource functions and proactively apply updates. To make appropriate decisions, the autonomic systems are equipped with learning capabilities.

Self-healing stands for the ability to detect, diagnose, and repair localised problems in hardware and software. Here, a possible method would be the online analysis of logfiles, possibly augmented with data from special monitoring components. If the system cannot solve the problem, it informs the administrator.

The **self-protection** of an autonomic system is accomplished by methods to defend the system against malicious attacks and failures not covered by the self-healing mechanisms.

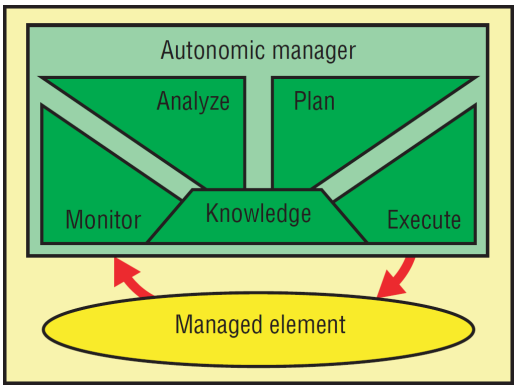


Figure 2.5: The model for an autonomic element as defined by IBM [105]

Kephart & Chess model an autonomic system as a set of interactive entities that contain resources and provide services to other entities and human users. Such an entity is called an *autonomic element*. Each autonomic element manages its own behaviour and interactions in accordance with policies that humans or other elements have set. The system-wide self-management will emerge from the local self-management of the elements and their interactions.

Figure 2.5 shows the model for an autonomic element. It consists of one or more managed elements and one autonomic manager. The **managed element** is the part that provides the service or a resource. This part can be a hardware component such as a printer, a storage device or a CPU, or a software resource such as a database or a name service. The autonomic element provides interfaces to be monitored and controlled by the autonomic manager. The **autonomic manager** monitors the environment and

the managed element. Based on the analysis of this information, the autonomic manager creates and executes plans to satisfy the four aspects of self-management. Here, the MAPE loop is extended by a shared pool of knowledge (MAPE-K), which all four steps can access. It contains all available data necessary to fulfil the self-management. That can include the system policies set by the administrator, information about the environment, historical data, or the results of the learning process.

In the sense of self-adaptation, the AC system model maps directly to Weyns' conceptual model. The managed element in an AC system is the managed system in Weyns' model and the autonomic manager in AC is Weyns' managing system. Furthermore, since an AC system consists of several interacting entities and has no centralised control instance (except the top-level policies), we see that an AC system fulfils our definition of self-organisation.

Autonomic Computing Applications

Starting from this sketchy vision, autonomic computing has received broad attention [118] in the last two decades. In 2003, *Koehler et al.* presented an AC architecture that ensures a specific behaviour [111]. *White et al.* defined architectural requirements for achieving the goals of AC in 2004. In the following years, many more architectures and models were proposed. An overview is given in [45].

On the practical side, autonomic computing designs were applied in several domains: adaptive webserver applications [255], autonomic grid applications (*AutoMate* [165]), and large server farms [35]. *Liu et al.* presented a programming framework for autonomic applications [128]. A short overview of AC-based applications can be found in [210]. A practical guide to the design and implementation of AC systems was given by *Lalanda et al.* [118].

2.3.2 Organic Computing

Started in 2003, the *Organic Computing* (OC) initiative [146] is the second major initiative dealing with SASO systems. While running parallel with AC, the two initiatives have different backgrounds: AC is rooted in commercial applications and centred in the US, while OC originates from an academic background in Europe.

The objective of OC is the technical usage of principles observed in natural systems. Against the background of increasingly complex computer systems, the central idea of OC is the orientation towards the needs of the humans who use and depend on such systems. To meet this idea, OC systems are inspired by the notions of independence, flexibility, and autonomy as basic properties of natural systems. Hence, OC defines an organic computing system as a self-organising, self-configuring, self-healing, self-protecting, self-explaining, and context-aware technical system that adapts dynamically to the current

conditions of its environment [147, 232]. In summary, an OC system should behave *life-like*.

In contrast to the authors of the *Vision of Autonomic Computing* [105], the initiators of OC give no closer explanation of what the self-* properties (e.g. self-healing) mean or what they should accomplish. Although for AC the intended goal of the self-* properties is described, formal definitions are missing. This gap is closed by *Berns et al.*, who give formal definitions of several self-x properties [20]. Using AC as a reference, the definitions are given in the general domain of self-adaptive and self-organising systems. This makes them applicable to OC systems if needed.

Another difference between AC and OC is the intended use. While AC is directed at IT infrastructure systems, OC focuses on large collections of intelligent devices providing services to humans [200]. Despite this broader spectrum of applications, most OC systems follow the basic *observer/controller* design pattern introduced in [146] and refined in [189] which is the core of most OC system models.

The Organic Computing System Model

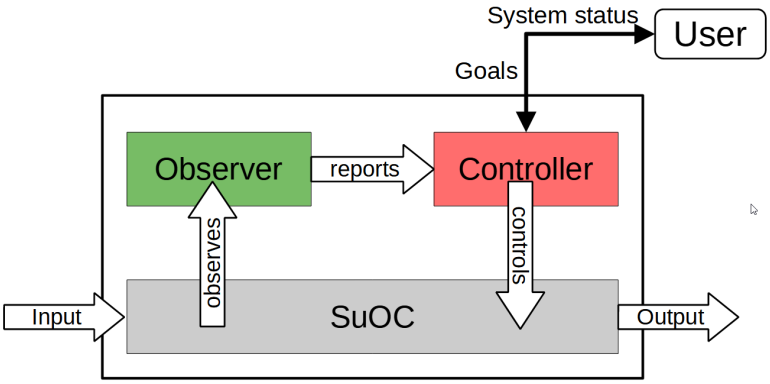


Figure 2.6: The basic concept of the Observer/Controller design pattern [189]

The orientation towards the needs of the human user requires OC systems to be able to react to changes. Not only those needs, given as system goals, might change at runtime, but also the environment and the internal conditions of the system can change in a way that prevents the system from fulfilling the needs/system goals if no action is taken. Such a reaction usually results in an adaptation of the system behaviour or conforms with one or more of the self-* properties. For this to be appropriate, the system needs a feedback loop. While in AC systems the feedback loop is designed as a MAPE-K loop, most OC systems use the more general approach of the *observer/controller* pattern. This architecture consists of three main components: the *observer*, the *controller* and the *system under observation and control* (see Figure 2.6).

The **system under Observation and Control (SuOC)** is, analogous to the managed element in AC, the productive part of the system. It performs the actions necessary to comply with the system goal. This capability is self-contained, i.e. the SuOC provides its functionality without the need for the observer and controller and will remain functional even if the controller or observer breaks down. The SuOC is generally considered a set of individual elements with specific attributes (in terms of a multi-agent system). The SuOC needs to provide interfaces for monitoring and receiving control commands.

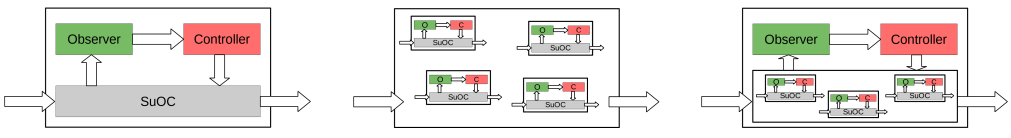
The **observer** monitors the state of the SuOC and its dynamics, including raw data, the SuOC elements, and global system attributes. Furthermore, it collects information about the environment, either directly or through the sensor data of the SuOC. The main tasks of the observer are the creation of an aggregated view of the current global system status and the prediction of the future status of the system.

Using the information provided by the observer, the **controller** decides whether the SuOC needs a controlling interference. The three main reasons to interfere are

1. to influence the system such that a desired behaviour emerges,
2. to inhibit a currently appearing unwanted behaviour and
3. to alter the system so that specific unwanted behaviours cannot occur in the future.

To achieve this, the controller can influence the decision rules of the elements and the connections and relationships between the elements (and thus the structure of the SuOC).

The OC architecture can be extended to a multi-level control hierarchy. In some applications, it is beneficial to split the entities in the SuOC into several groups and assign to each group an intermediate observer/controller. These groups then act as entities of a higher level SuOC with its own observer and controller. An example of such a delegation of control is a traffic control system where each intersection is modelled as an OC system to control the local traffic flow, and groups of several intersections are controlled by a higher-order observer/controller to optimise the regional traffic flow [27]. Figure 2.7 shows some of the possible control structures. In a hierarchical control structure, mixtures of central and distributed OC systems can be implemented in the lower control levels. The OC architecture can be extended and customised in several other ways. We refer to [227, 229] for more examples.



(a) central (b) distributed (c) hierarchical

Figure 2.7: Some possibilities of the control structure in OC systems

Compared to *Weyns'* self-adaptation model, the SuOC is the managed system in Weyns' model, and the tandem of Observer and Controller in OC maps to Weyns' managing system. Regarding self-organisation, we have to note that an OC system does not fulfil *Elmenreich & Meer's* definition (see definition 2.2.3). Although often implemented as a collection of several entities, the SuOC can be a single system which, furthermore, is not free from a central controlling instance. On the other hand, within the OC initiative the OC systems are defined as self-organising.

Organic Computing Applications

Several applications use the observer/controller design pattern or are inspired by it: the *Organic Computing in Off-highway Machines* [250] proposes a system to manage the components of off-highway machines (e.g. tractors) to increase efficiency. The OC design has been proposed for small devices such as smart surveillance cameras [225] and simple cleaning robots [188], in a complete smart home environment [12], in a regional traffic control system [176], for network protocols [223], and for middleware task allocation with artificial hormone systems [160]. An overview of OC systems can be found in [229].

2.3.3 Multi-Agent Systems

A *multi-agent system* (MAS) is a set of multiple interacting agents [247]. While *Woolridge* defines an *agent* as a computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its delegated objectives [247], other authors are less restrictive and accept every type of autonomous entity as an agent, even humans [57]. Although most applications only use one type of agent, heterogeneous MAS are possible [121], such as teams consisting of robots and humans. Another definition with a focus on interaction is given in [239]. There, an MAS is defined as a system comprising two or more agents, cooperating with each other while achieving local goals.

Besides autonomy, agents in an MAS have two other important properties: 1) they only have limited information about the internal states and environmental conditions of other agents. And 2) there is no central instance that controls the agents. Both total knowledge and central control would allow the agents to fully synchronise, which would reduce the MAS to a monolithic system [163].

Multi-agent systems are used to model or solve complex problems where a monolithic system design would be inefficient or even foredoomed. Another area of application are heuristic approaches to problems that are either not solvable or too complex to be reasonably solved with other means [98].

In contrast to AC and OC, which are approaches to the system engineering of SASO systems, multi-agent systems provide a way for modelling and analysing specific problem

domains.

In general, self-organisation or self-adaptation are no required properties of multi-agent systems. A simple example of an MAS that does not require them is Conway's Game of Life (see Chapter 2.2.3). Here, the properties are not defined in the design. Yet, the self-organisation emerges at runtime. Of course, MAS with these properties are important tools in research and application [48, 242].

Multi-Agent System Architectures

In literature, multi-agent systems are differentiated by the types of their agents. The main classification aspect is how an agent makes decisions on its actions. *Woolridge* [247] distinguishes between *table-driven*, *reactive*, *deductive reasoning* and *practical reasoning agents*.

Table-driven agents can be used if the environment can be modelled to be discreet, i.e. with a finite number of variables and a finite number of states. Here, the agent can be equipped with a lookup table that contains the desired action for all possible states of the environment [217]. Besides the discreet environment, these agents are limited by the practical aspects of the table (available memory/space and lookup time).

The terms **reactive** and **reflex(ive) agents** are used synonymously and describe agents that use a set of rules to determine their next step [247]. In a classical IF-THEN format, each rule maps a specific condition of the environment and the agent to an action. Although these agents are not limited to discreet environments, they can be seen as a subtype of table-driven agents.

Deductive reasoning agents use logical deduction [192] to determine their next step. The information on the environment, the internal state, and the desired behaviour are stored as formulae of first-order predicate logic [51]. The agent then uses logical deduction rules to transform its current state and the available information into an action [247]. Here, the focus is on the available information. Since the information is only based on the agent's perception, it has no indication of whether the information is actually accurate. Therefore, the information is often called the *beliefs* of the agent.

The class of **practical reasoning agents** sets the focus on the eventual action. The notion of practical reasoning is defined by *Bratman* as the matter of weighing conflicting considerations for and against competing options, where the relevant considerations are provided by what the agent desires/values/cares about and what the agent believes [24]. Practical reasoning is divided into two parts: *deliberation* and *means-ends reasoning*. Here, deliberation is the act of deciding what the agent wants to achieve and means-ends reasoning is the act of deciding how to achieve that. A prominent instance of this approach are **BDI agents** [183]. The BDI agent model separates three entities: *beliefs*, *desires*, and *intentions*. A desire is a compromise between the (sometimes conflicting)

goals that the agent wants to achieve. An intention is the outcome of the beliefs and a desire. It is the basis for the agent's next actions. Figure 2.8 illustrates the concept.

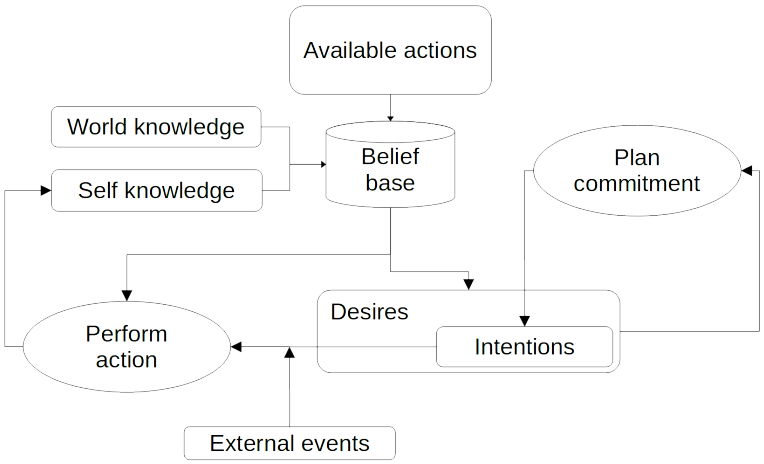


Figure 2.8: A model for BDI agents [1].

In the research domain of *artificial intelligence*, agents are usually classified with the system given by *Russel & Norvig* [197] which aims at the level of perceived intelligence and capabilities and contains five classes:

1. **Reflex agents**.
2. **Model-based agents**, that use the perceptions of the environment, the current internal state, a set of condition-action rules, and a model about the environment to determine the next action.
3. **Goal-based agents** are model-based agents with a goal function that guides their actions.
4. **Utility-based agents** are goal-based agents that are trying to maximise the utility in the goal selection.
5. **Learning agents** can learn from their past actions and the environment.

Multi-Agent System Applications

Multi-agent systems have been implemented in a wide range of industrial and commercial contexts. *Luck et al.* list the following domains with MAS applications: manufacturing, process control, telecommunication systems, air traffic control, traffic and transportation management, information filtering and gathering, electronic commerce, business process management, human capital management, skills management, (mobile) workforce management, defence, entertainment, and medical care [131]. Examples from these domains include a system for managing a dynamic business process for British Telecom [98], a system for testing negotiation protocols in e-commerce [158], and a decision support system

for managing electricity transportation systems [99].

Another application for MAS is the modelling and simulation of interacting entities. Here, examples are the simulation of predator-prey systems [123], the modelling of social interactions of humans [136], and simulating flocks of birds [218].

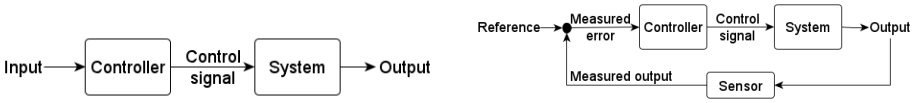
Creating MAS simulations and implementing software agents take some effort. Several frameworks and simulation tools have been developed for a wide range of programming languages to simplify these tasks. An overview is given in [161]. One of these frameworks is MASON [132], a versatile Java framework for multi-agent simulations. The evaluations (cf. Chapter 5) in this thesis are implemented with it. Moreover, several specialised programming languages for implementing agents have been created. A survey on such languages can be found in [30].

2.3.4 Further Initiatives and Research Domains

The concepts of self-adaptation and self-organisation in technical systems are not limited to the three domains mentioned above (AC, OC and MAS). One crucial building block for self-adaptation is the concept of feedback loops, which are studied in *control theory*. The research field of *cybernetics* can be seen as a predecessor of AC and OC. Other related domains are *pervasive/ubiquitous computing, proactive computing, complex adaptive systems,* and *interwoven systems*. Let us have a short glance at them.

Control theory

As a field in engineering and mathematics, control theory deals with the control of dynamic systems. While early applications of feedback control can already be found in antiquity (e.g. the water clock by Ctesibius of Alexandria [262]), the theoretical foundations were laid by *Maxwell* in 1868 [138]. The objective of control theory is to create models or algorithms that steer the input of a system to reach a desired state or output of the system while maintaining stability of control. The core concept to achieve this are control loops where the system input is controlled by a controller instance. These loops are typically distinguished into open and closed loop systems [69]. In closed control loops (see Figure 2.9b), the system output is measured and compared to a reference value. The deviation is used by the controller to create a control signal, which is used as the system input. An example of such a closed-loop system is a heating element with a thermostat. The thermostat measures the room temperature and controls the flow of warm water to the heating element. This flow then influences the room temperature. The feedback loop in Weyns' self-adaptive system model is an example of such a closed control loop. Figure 2.9a shows the concept of the open loop. The output of the system is not considered by the controller. Here, an example is a heating controlled by a timer where the controller determines the flow of warm water only by the current time.



(a) open control loop (b) closed control loop

Figure 2.9: The two principal types of control loops in control theory

Cybernetics

Cybernetics is a wide-ranging field across several research domains that deals with control and regulation in technical and non-technical systems [7]. The term *cybernetics* was introduced by *Wiener* in the late 1940s [243]. Contributions to cybernetics research come from diverse domains, including biology [139], sociology [202], medicine [113], and, of course, engineering [62]. The core concept of cybernetics is feedback – the observed results of actions are taken as input for further actions. This is generally modelled with a closed control loop as defined in control theory. Also, *Ashby* presented his ideas on self-organisation [8] in the context of cybernetics.

Pervasive/Ubiquitous Computing

In the past, the actual physical location where computation took place was easy to tell: the mainframe computer in the big building across the campus or the desktop system in front of you. Today, this is more challenging. We have numerous small and interconnected devices that either perform the computation or delegate it through the network to another system. Computation takes place everywhere and at all times. The research fields of pervasive computing [87] and ubiquitous computing deal [184] with this phenomenon. Although the two terms are often used synonymously, some authors see slight differences in their meaning [191]. The combination of ideas from pervasive and autonomic computing is used to address complexity and management issues in distributed systems [80]. And since pervasive computing investigates systems with many computing elements, the connection to multi-agent systems is at hand [3].

Proactive Computing

The main focus of proactive computing [220] is to proactively anticipate the needs of a user. The ideas of autonomic and organic computing were inspired by proactive computing. For example, autonomic and proactive computing share concepts from ubiquitous computing and distributed systems but differ in their primary goals: management and interactivity in autonomic computing and anticipation in proactive computing [237].

Complex Adaptive Systems

The research field of complex adaptive systems (CAS) investigates multi-agent systems where the agents and the system itself have self-adaptation capabilities. Important properties of CAS are communication, cooperation, specialisation, and self-organisation [63]. AC and OC systems are subclasses of CAS with additional properties.

Interwoven Systems

Interwoven Systems is a research initiative that studies complex systems consisting of several subsystems that show specific properties like mutual influence, heterogeneity, uncertainty of behaviour, and real-time reactions. The focus of the initiative, which emerged from early contributions to the organic computing initiative [224], is to provide solutions for questions of manageability, self-adaption, self-organisation, and (self-)integration of such systems [17]. Here, self-integration, i.e. the integration of new subsystems into the existing system, is the primary objective [16]. Therefore, the goals of the Interwoven Systems initiative can be seen as an extension of the OC research focus. This perspective yielded an OC-based reference architecture for Interwoven Systems [231].

Self-aware Computing

Self-aware Computing deals with systems that learn models by capturing knowledge about themselves and their environment and use these models to reason and act [114]. While the general concept of self-awareness can be found in many research fields (see [125] for an overview), this initiative puts the focus on the knowledge-capturing models and the usage of the knowledge.

2.4 System model

The ideas presented in this thesis do not require a very special system model. They focus on the system engineering aspects, i.e. the design and operation of SASO systems and not specific system concepts or models. Hence, we will use a formal model that uses the fundamental ideas of AC, OC and MAS and thus is basically valid in all these environments. However, before we start, we need to specify a few terms.

2.4.1 Definitions for the System Model

Autonomous System

The terms **autonomous systems** and **autonomy** (in technical systems) are not uniformly defined in literature and have seen some changes over time. In 1978, *Sheridan &*

Verplank defined autonomy as *"the computer doing [the] whole job if it decides it should be done, and if so, tells the human if it decides he should be told"* [206]. After this, authors reduced autonomy to the absence of human intervention [14]. Later, scientists refined this approach and added several properties to suit their needs. One notable property that was added is adaptability in the sense that an autonomous system can adapt and change its behaviour to reach its goals [97]. An overview of several definitions of autonomy and their historical development is given in [151].

Since we will separately require the self-adaptation capability in our system model, we will use the definition from *Beer et al.*: autonomy is the ability to perform given tasks based on the system's perception without human intervention [14].

Goal Function & Utility Function

The terms **goal** and **goal function** are rather general concepts about the purpose of a system rather than a strictly definable notion. *Bossel* describes goals as properties that act as landmarks, propensities, or attractors guiding system evolution and development. When quantified and used in models, they are referred to as goal functions [23]. For multi-agent systems, *Russell & Norvig* define goal functions as functions that describe desirable states of the system and give an example: for a taxi, having arrived at the passenger's destination is a desirable state. Furthermore, they define **utility functions** as functions that assign a numerical value to each possible state of the system. Therefore, the system can find optimal states or sequences of optimal states. For the taxi, that means not only arriving at the destination but also doing so in an optimal way (regarding e.g. time, travelled distance, price) [197]. A more formal definition of a utility function comes from economics. If given several options to choose from, an individual will order these options according to personal preference under the aspect of value or utility gained from each option. A utility function is a continuous function from these options to the real numbers that preserves this order [44].

Self-Awareness

The notion of **self-awareness** is a building block of self-adaptation [117]. In the same way that there are different definitions of self-adaptation, there are different ideas on how to define this notion. For *Hinchey & Sterritt*, self-awareness is the ability of a system to be able to monitor its resources, state, and own behaviour [92]. A more formal definition of self-aware computing systems is given by *Kounev et al.*: Self-aware computing systems are computing systems that learn models by capturing knowledge about themselves and their environment (such as their structure, design, state, possible actions, and runtime behaviour) on an ongoing basis and reason using the models (e.g. predict, analyse, consider, and plan) enabling them to act based on their knowledge and

reasoning (e.g. explore, explain, report, suggest, self-adapt, or impact their environment) in accordance with higher-level goals, which may also be subject to change [114].

In their survey on self-awareness in computing systems, *Lewis et al.* have compiled several more definitions [124]. Here and in [114], an overview of applications in the several research fields of computer science is given.

For this thesis, we will define *self-awareness* as the ability to monitor and analyse all internal and external parameters that have an influence on the system.

2.4.2 The System Model

We model a SASO system S as a collection $A := \{a_i, i \in \mathbb{N}\}$ of autonomous subsystems a_i that are able to adapt their behaviour based on self-awareness of the internal and external conditions. We further assume that such a subsystem is an entity that interacts with other entities, i.e. other systems, including hardware, software, humans, and the physical world. Everything perceived via external sensors, including these other entities, is referred to as the *environment*. Each $a_i \in A$ is equipped with sensors and actuators (both physical and virtual). Internally, each a_i consists of two parts: the productive system part *PS*, which is responsible for the primary purpose of the system, and the control mechanism *CM*, which controls the behaviour of the PS (i.e. performs self-adaptation) and decides about relations to other subsystems. Each subsystem has its own local configuration and provides read access to it.

This configuration is realised using a vector c_i. It contains a specific entry for each control variable that can be altered to steer the behaviour. Independent of the logical realisation of a particular parameter (e.g. as real value, boolean/flag, integer or categorical variable), the entries in the vector are numeric values. Each subsystem has its own configuration space, i.e. an n-dimensional space defining all possible realisations of the configuration vector. The combination of the current configuration vectors of all contained subsystems of the overall system S defines the joint configuration of S. We assume that modifications of the configuration vectors are done by the different CM only, i.e. locally at each subsystem, and are the result of the self-adaptation process of the CM.

Figure 2.10 illustrates this concept with its input and output relations. The user describes the system purpose by providing a utility or goal function U, which determines the behaviour of the subsystem, and then takes no further action to influence those decisions. The actual decisions are taken by the productive system and the CM based on the external and internal conditions and messages exchanged with other subsystems.

In comparison to other system models, the distinction between the CM and the PS corresponds to the separation of concerns between the *System under Observation and Control* (SuOC) and *Observer/Controller* tandem [229] in the terminology of Organic

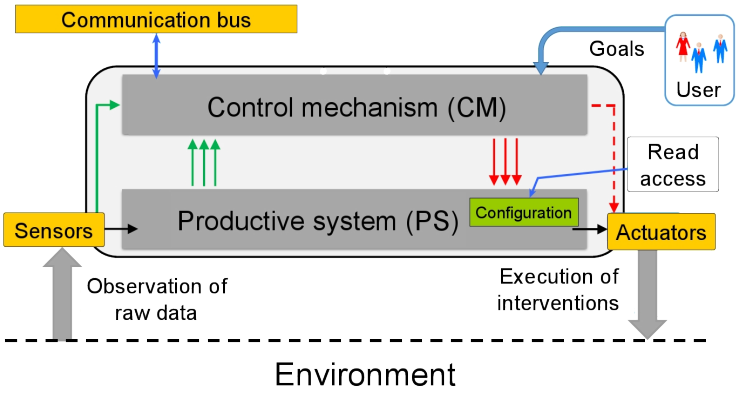


Figure 2.10: Schematic illustration of a single subsystem of the system model. The green arrows show observation flows; the red arrows depict control flows.

Computing [232], the *Managed Resource* and *Autonomic Manager* in terms of Autonomic Computing [105] or the *Managed System* and *Managing System* by Weyns. As in AC and OC, the system behaviour is not given by direct user commands but by user-defined goals. Furthermore, since we have a collection of multiple interacting subsystems, our SASO system is a multi-agent system. If the system constellation is, in principle, open and may contain heterogeneous systems, it is even an Interwoven System.

On the side of differences, we note that while AC requires the MAPE-K cycle in the controlling instance, our CM can be implemented with any controlling concept. And while the CM is modelled as a single component in OC, the observer and controller are separated.

In general, the control mechanism is not necessarily limited to supervising a single productive system. In the same way OC systems can be modelled with a multi-level control structure, a single CM in our system model can control several independent productive systems or several lower-level CM/PS tandems (cf. Figure 2.7).

For this thesis, the actual control structure is less important. Our approach to the assessment of the self-adaptation behaviour is solely based on the configurations of the productive systems. In distributed systems, the behavioural assessment will be done by an external instance. In central or multi-level systems, the assessment can be implemented in the top-level CM. Here, the assessment results can even become part of the self-adaptation process.

Chapter 3

State of the Art

A crucial part of understanding how SASO systems behave is to quantify and evaluate their properties. Over the years, several metrics and frameworks have been proposed to aid such an evaluation. A survey on evaluation approaches of SASO systems is presented in [65]. For an extensive survey on evaluation approaches (including methods, metrics and benchmarks) in the domain of Self-aware Computing, we refer to [83].

We will take a tour through the literature regarding these topics with a focus on run-time metrics for self-adaptation and self-organisation and will emphasise the need for an integrated system-wide consideration of a measurement framework for SASO properties. Of course, we are not the first to do so. Recent overviews on metrics – with a slightly different focus – are given by *Raibulet et al.* [179] and *Eberhardinger et al.* [52].

3.1 Metrics for Self-Adaptation

The contributions to the evaluation of self-adaptation usually focus on specific properties of the adaptation process:

- *Villegas et al.* separate these properties into the categories *performance* of the adaptation process, *dependability*, *security*, and *safety* and state the specific metrics apply to one or more of these categories [235].
- *Kaddoum et al.* present several metrics to evaluate adaptive systems based on their architecture, the complexity of their algorithms, and durations of operations during runtime [102].
- For software systems, *Raibulet & Masciadri* propose three sets of metrics based on the structural and architectural aspects of adaptivity, the costs of adaptations and the interactions of the user with the system [180].
- *Eberhardinger et al.*, on the other hand, distinguish the categories *time-oriented*

metrics and *solution-quality-oriented metrics* [52].

Before we give a short overview of metrics related to self-adaptation, we should mention that in literature the term *adaptability* occurs in our context quite often. Usually, it refers to the notion of the general ability of a system to perform adaptation. However, when it comes to metrics, the terms self-adaptation and adaptability sometimes become ambiguous. There are, of course, several metrics that try to measure adaptability. In the following, we will only include metrics that are applicable in a wider range of systems and focus on the self-adaptation process itself and therefore aim towards our requirements.

3.1.1 Working/Adaptation Time Metric

For their runtime metrics, *Kaddoum et al.* assume that phases with and without self-* operations can be distinguished. They then define the Working/Adaptation Time Metric WAT as $WAT = (\text{time spent on working})/(\text{time spent on adaptation})$. In the same way, they define metrics based on the ratio of response time, the quality of response, the CPU load, and the communication load during normal and self-* phases [102]. The WAT metric has been investigated in detail by *Camara et al.* for software architecture scenarios [28].

3.1.2 (Un-)Availability

Candea et al. present another metric based on distinguishable states. They borrow the notions of *mean time to recover (MTTR)* and *mean time to fail (MTTF)* from the domain of system maintenance, where the latter is the average time a system runs before a failure occurs and the former is the average time until the system is recovered after a failure. Assuming that a system during a self-* operation is considered recovering, they define the availability A as $A = MTTF/(MTTF + MTTR)$ and the unavailability U accordingly as $U = MTTR/(MTTF + MTTR)$ [29].

3.1.3 Situation Performance

The Situation Performance, as proposed by *Taranu & Tiemann* [219], is a metric focused on the quality of the outcome of the adaptation process. Here, a situation is defined as a problem for the system to solve or a condition to adapt to. *Taranu & Tiemann* assume that such situations can be decomposed into sub-situations and that each adaptation to a sub-situation can be assigned a value determining the cost C_{subsit} or effort. With C_{max} defined as the maximum cost of all the costs for the sub-situations, the Situation Performance SP for a situation sit is defined as

$$SP(sit) = 1 - (\sum_{subsit \in sit} C_{subsit})/(\sum_{subsit \in sit} C_{max}). \tag{3.1}$$

3.1.4 Requirement Performance

Becker et al. evaluate adaptive systems by their ability to satisfy requirements based on the prerequisite that the (dis-)satisfaction of a requirement can be measured [13]. An example would be the requirement R for a system to keep its temperature below a specific maximum temperature R_{max}. If $\Delta(R)$ denotes a function describing how dissatisfied the requirement is during a given time frame, then the Requirement Performance m_R is defined as

$$m_R = \begin{cases} 0, & \text{if } \Delta(R) \geq R_{max} \\ 1 - \Delta(R)/R_{max} & \text{otherwise.} \end{cases} \tag{3.2}$$

3.1.5 Decision Benefit

This metric, defined by *Reinecke et al.* [185], requires that the benefit from every decision that a system makes can be measured. With p_i as the benefit for the decisions taken at time step i, three sets of time steps are defined: $D_+ = \{i | p_{i-1} < p_i\}$ is the set of steps where the decision has a bigger benefit than in the step before, $D_- = \{i | p_{i-1} > p_i\}$ are those with a smaller one, $D_= = \{i | p_{i-1} = p_i\}$ are those with an equal benefit. Then, with $\Delta_i = (p_i + p_{i-1})/2$ and N as the total number of time steps, the Decision Benefit metric DB is defined as

$$DB = \frac{\sum_{i \in D_+} \Delta_i + \sum_{i \in D_=} p_i}{N - 1}. \tag{3.3}$$

Raibulet et al. [179] call this metric *Adaptivity Metric* in their overview paper on metrics.

A similar idea, namely a cost-benefit analysis, is used by *Gerostathopoulos et al.* to evaluate which assessment strategy should be used in a specific situation [64].

3.1.6 Population Metric

For *Castiglioni et al.*, a large-scale system shows adaptability if it is able to reach a desired target state (or close to it) over time, even if the initial conditions are changed. To measure this ability, they first define a distance metric on the set of system states, and then they lift this metric to probabilistic distributions (over the set of system states) using the Wasserstein metric [149]. Finally, they extend this metric to define a distance metric between two system states and a set of observation times, which they call the Population Metric [31].

3.1.7 Discussion of the Metrics

As stated in Chapter 1.3, we are interested in metrics that can be applied to a wide range of systems and domains and meet the following requirements:

- **Req1:** They do not require expert knowledge. They can be implemented without knowledge about the system goal, the application domain and other aspects of expert knowledge.
- **Req2:** They do not require knowledge about internal system states.
- **Req3:** They do not require knowledge about the structure of the system or the dependencies of its components.
- **Req4:** They do not require knowledge about the outcome of self-adaptation processes.
- **Req5:** They can be evaluated at runtime. The metrics identify unusual events with no or an acceptable delay.

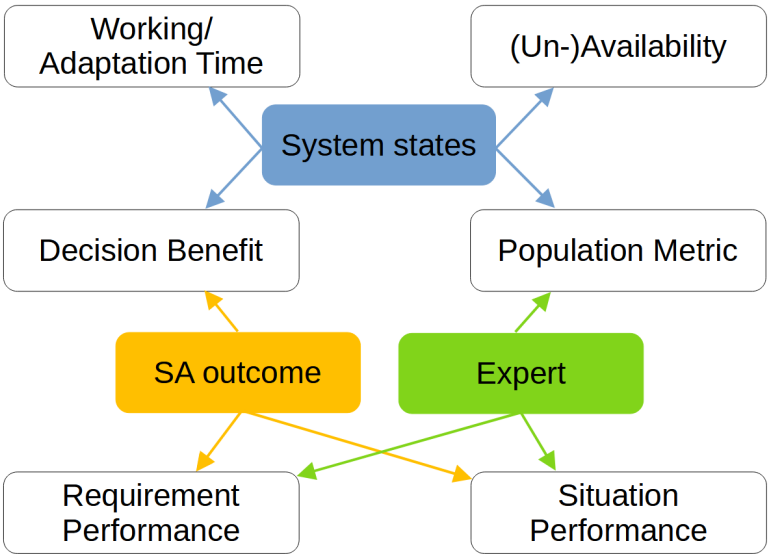


Figure 3.1: Illustration of the requirements for the existing self-adaptation metrics. They require **expert** knowledge (contradicting **Req1**), knowledge about the internal **system states** (contradicting **Req2**) or the **outcome** of the self-adaptation (contradicting **Req4**).

When we compare the given metrics with the requirements, we see every metric does not comply with at least one of the requirements, pictured in Figure 3.1: The *Working/Adaptation Time Metric* and the *(Un-)Availability* require distinguishable states of working and adapting. They are not applicable in systems where adaptations are made during working time or in distributed systems where only some parts undergo adaptation. The *Situation Performance* and the *Decision Benefit* require the definition of a cost or benefit function during design time. To give useful insight into the adaptation, such a function would be tailored to the specific system or at least to the domain of operation. In the same way, the *Requirement Performance* depends on a system-specific function to

evaluate the satisfaction of a specific requirement. Using the *Population Metric* as a tool for runtime assessment requires the definition of the distance metric of states at design time and the definition of desired target states.

As we can see, the metrics contradict our postulated requirements **Req1**, **Req2** and/or **Req4**.

3.2 Metrics for Self-Organisation

Besides adaptation of behaviour, measuring self-organisation (SO) is of particular interest since self-organisation refers to an adaptation of the system's structure (e.g. in terms of connections, interactions, or relations between subsystems).

In their effort to compare metrics for self-organisation, *Birdsey et al.* [21] state that existing metrics measure three properties of self-organisation: *emergence*, *stability*, and *criticality*. Furthermore, they note that the aspect of *adaptability* can be seen as a necessary prerequisite for self-organisation or as a sub-process of self-organisation.

For the scope of this thesis, we are interested in things that happen at runtime. Thus, let us take a closer look at metrics suitable for this purpose.

3.2.1 Interaction Metrics

The Interaction Metrics are a set of metrics defined by *Chan* [34] for agent-based simulations to measure emergence. The core of the metrics is the number of interactions an agent has with its neighbours at a given time. If an interaction leads to a state change for an agent, it is called an effective interaction. From this, *Chan* calculates the total number of state changes I_t in the system at a given time t, the cumulative number of state changes X_{it} for an agent i for all time steps until t, the normalised cumulative number Y_{it} of state changes $Y_{it} = X_{it}/max(X_{it})$, and the total cumulative number of state changes $Z_t = \sum_i X_{it}$. Chan states that an emergent behaviour arises if the time series of the calculated values deviate from normality. *Birdsey et al.* note that not all of these time series are useful in all scenarios.

3.2.2 Graph-theoretic Metrics

Based on the interactions, *Birdsey et al.* [21] evaluate metrics based on the interaction graph. Here, the nodes of the graph are the agents, and they are connected if an interaction takes place in a given time frame. They then look at graph-theoretical properties such as the graph density or the number of connected components [240]. Comparable approaches were presented by *Kantert et al.* [103], *Rudolph et al.* [193], and *Tomforde et al.* [226].

3.2.3 Limited Bandwidth Recognition

This metric was proposed by *Brown & Goodrich* for behaviour distinction in swarms [26]. They use a Bayesian classifier to calculate the probabilities that the system exhibits a particular collective behaviour based on the features of the agents (such as velocity or orientation). Limited bandwidth recognition can be used to distinguish between various types of the same emergent behaviour, e.g. birds behaving collectively by flocking closely together or following each other in a toroidal structure.

3.2.4 Oscillation Detection

The Oscillation Detection metric $OD(t)$ was created by *Birdsey et al.* [21] and describes whether the state $S(t)$ of the system at time t has occurred before. Suppose there is a time point $t - k$ such that $S(t) = S(t - k)$ and k minimal, then $OD(t) = k$ and 0 otherwise. In deterministic systems, this metric shows oscillation and can be used to detect emergence and adaptivity.

3.2.5 System Complexity

For the System Complexity metric, *Birdsey et al.* adapt a metric presented by *Deaton & Allen* [43]. They state that the complexity of a system can be determined by the change in the number of interactions at different scales between two consecutive time steps. For example, the complexity of a distributed system is determined by the interactions between software components, the interactions between hardware components, and the interactions that take place over the network. They define System Complexity SC as $SC = |I(T_x)|/|I(T_{x-k})|$ where $I(T_x)$ is the number of interactions that took place at time x, and k is a window size [21].

In Chapter 2.1, we saw that the notion of *complexity* has a broad spectrum. Therefore, it is no surprise that there is a lot of literature with approaches to measure their respective idea of complexity. *Birdsey et al.* are listed here because their approach of using interactions is related to the Interaction Metric and the graph-theoretic approaches. For more on complexity measures, we refer to [244].

3.2.6 Entropy

Starting from the idea [66] that system states with increased organisation represent lower information Shannon's Entropy $H(X) = -\sum_{x \in X} p(x)log(p(x))$ [40] and derived metrics have been used to measure self-organisation. Usually, X is the (discrete) set of system states [166], but other sources were investigated. *Tomforde et al.* [226] use the communications messages and apply the Kullback-Leibler divergence measure (which

is derived from Shannon's entropy). We will come back to this specific measure in Chapter 4.3.8. Other metrics based on entropy used to investigate self-organisation are Mutual Information [38, 233] and Transfer Entropy [201, 233]. *Wright et al.* present an entropy measure for continuous simulations of systems with high degrees of freedom based on a function of an attractor's dimension within the underlying state space [248]. *King & Peterson* propose a self-organisation metric where the entropy is applied to local neighbourhoods [106].

An overview of several usages of the general concept of entropy apart from quantification in the domain of self-organisation is given in [175].

3.2.7 SO Performance

Eberhardinger et al. present a metric to evaluate the performance of distributed self-organisation mechanisms in a system. For them, the SO Performance is a weighted sum of the time-performance and the quality-performance. For the time-performance, first the fraction of time it needs to reconfigure is calculated for every agent. Then, for a single evaluation run, the average over all agents of these times is taken. Finally, the time-performance for the system is the average over several evaluation runs. The quality-performance is again the average over all agents and several runs over a domain-specific quality-function that evaluates the outcome of the self-organisation process [53].

3.2.8 Discussion of the Metrics

How do the self-organisation metrics fit into our framework? Let us repeat our requirements for metrics in the framework:

- **Req1:** They do not require expert knowledge. They can be implemented without knowledge about the system goal, the application domain and other aspects of expert knowledge.
- **Req2:** They do not require knowledge about internal system states.
- **Req3:** They do not require knowledge about the structure of the system or the dependencies of its components.
- **Req4:** They do not require knowledge about the outcome of self-adaptation processes.
- **Req5:** They can be evaluated at runtime. The metrics identify unusual events with no or an acceptable delay.

The *Limited Bandwidth Recognition* requires sufficient training data and an expert to select the appropriate features. The interaction metrics require an upfront definition of what an interaction actually is. The *Oscillation Detection* and the *Entropy* metrics require knowledge about system states. If we associate configurations with system states,

the *Oscillation Detection* could be used in our framework, but the metric aims to detect cycles of states. A disturbance will most likely fall into a phase of very small cycles. The meaningfulness of the metric is questionable in this case.

The *Performance* metrics are computed after several runs of the system. They can give no insight at specific points in time.

Generally, for the self-organisation and the emergence metrics, we can say that they try to measure a specific property or outcome of the self-adaptation process. The answer to the question of how much insight this gives into the adaptation process is system-specific.

Figure 3.2 summarises the requirements of the metrics. As we can see, the metrics contradict our postulated requirements **Req1**, **Req2 Req3** and/or **Req5**.

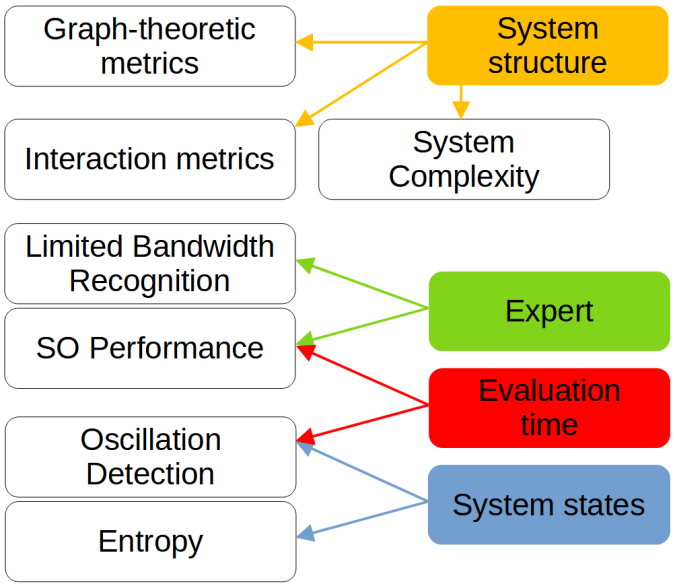


Figure 3.2: Illustration of the requirements for the existing self-organisation metrics. They require **expert** knowledge (contradicting **Req1**), knowledge about the internal **system states** (contradicting **Req2**) or the **system structure** (contradicting **Req3**), or they need longer periods of **time** to be evaluated (contradicting **Req5**).

3.3 Measuring Emergence

As outlined earlier, emergence is closely related to self-organisation. Therefore, measurements for emergence in the context of self-organisation have been investigated. One example is given by *Shalizi*, who presented an approach that is based on the efficiency of predictions and makes use of approaches known from the domain of information the-

ory [205]. A closely related approach has been presented by *Prokopenko et al.* in [174]. In general, both aim at assessing which level (i.e. from micro to macro) is more efficient for predicting the next states.

Similarly, *Holzer & De Meer* compare the information at the system level (i.e. the sum of all edges in a network-based representation) with the information of a lower level (i.e. a single edge in a network-based representation) [93]. As a result, emergence values are high for those systems with many components depending on each other and low for those systems that mostly comprise independent components.

O'Toole et al. propose the Decentralised Emergence Detection metric, which tries to detect emergence at runtime [159]. Here, a random number of agents calculate the statistical correlation between internal and observed variables in the environment for a given time frame. If the agents detect a change in the correlation, they use a consensus algorithm to determine if enough agents have experienced such a change and consider the situation as the result of a system-wide emergence event.

As an alternative, the Organic Computing domain used Shannon's discrete entropy measure as a basis for defining emergence. *Mnif & Müller-Schloer* modelled all relevant attributes of the system as random variables and determined emergence as a decrease of entropy based on self-organising processes [143]. This has been extended towards continuous attributes in [56] by deriving probability distributions of attributes and comparing them using divergence measures.

Within the context of emergence and self-organisation, the notion of *homeostasis* is used as another metric. *Gershenson et al.* define their homeostasis metric as $H := 1 - d(I_{in}, I_{out})$ where I_{in} is the system's state before an adaptation, I_{out} the state afterwards and d the Hamming distance of the two states [66]. After choosing a suitable string representation of the states (e.g. as a binary string), d measures the fraction of changed symbols in that string. However, finding a suitable representation for configurations is challenging: while the numeric difference between 1.0000 and 0.9999 and between 1.0000 and 1.0001 is identical, the Hamming distance is different.

In conclusion, we can say that emergence has experienced a wide range of approaches for measurement. For our framework, all of the mentioned ideas require too much system knowledge.

3.4 General System Performance Metrics

When SASO systems are deployed to production, monitoring their general performance is needed to identify their degradation or signs of upcoming failures. Although performance is not a SASO property, monitoring performance indicators together with applying our framework helps to identify abnormal events. Changes in performance could be the cause

or the result of an adaptation process.

Meyer presents a metric that combines performance and reliability and calls it *Performability*. It is based on the idea that performance and reliability can be modelled as random variables. The Performability is then a function that maps performance levels to the probability that the system performs at that level under given conditions [141]. *Meyer's* framework applies to all systems with a possible degradation of performance and is not limited to SASO systems. *Silva et al.* give an overview of general performance metrics (e.g. response time) that were applied to self-adaptive systems [41]. None of the presented metrics there are intended to give insight into the self-adaptation process. *Porter et al.* present a performance evaluation test-bed for self-adaptive and self-healing systems called TESS [172], which calculates metrics from the logs generated at runtime. For complex systems with heterogeneous components, *Das et al.* propose a runtime performance evaluation based on time series analysis of the measurable outputs of the single components [42].

Mandrioli & Maggio propose a method to test the performance of the self-adaptation layer of adaptive software systems, which provides probabilistic guarantees on the used metrics [135].

For two reasons, these performance metrics are no candidates for our measurement framework: they depend on a specific definition of performance and – more importantly – they do not use configurations as input.

3.5 Other Assessment Approaches

Besides evaluating metrics, other approaches to understanding the behaviour of SASO systems have been proposed.

For the field of Self-aware Computing, *Götz et al.* have compiled an extensive survey of approaches to evaluate such systems, including methods, metrics and benchmarks [83].

Naqvi et al. present a formal framework that assesses the system's behaviour by introducing perturbations (such as a turned-off sensor) and evaluating how the system reacts [154]. For rule-based self-adaptation, *Klös et al.* present an approach to behavioural analysis by constructing and evaluating formal models on the rules [108]. In the context of cybersecurity, *Musliner et al.* assess the quality of self-adaptation based on the number of predefined tests that the system passes after the adaptation [148]. A similar approach is presented by *Wuttke et al.*, who assess and compare self-adaptation strategies by implementing them in a defined scenario with given goals and conditions. In their case, that is a traffic routing simulation [251]. These approaches are focused on simulations and the design phase of the system. They are not designed to give insights into actual events during runtime.

Miner et al. investigated ways to assess the adaptability of a system of systems (SOS) like nationwide energy grids or the US Post Office [142]. They state that there is currently no metric that works for such a heterogeneous, large-scale system and argue that several domain-specific metrics need to be combined. On this basis, they define the System of Systems Adaptability Index $SoSAI$ over a set X of hand-picked metrics as $SoSAI = \sum_i \omega_i(X_i - \mu_i)/(\sigma_i)$. Here, X_i denotes the value of the Metric i, μ_i and $sigma_i$ are the mean and the standard deviation for the values of X_i over several evaluation runs, and ω_i is a weight for that metric. All the metrics need to be scaled to take values ranging from -1 to 1. Here, the domain-specificness distinguishes this approach from our framework and the fact that our framework aims to identify events at runtime.

Finally, we need to mention the field of validation and verification of self-adaptive systems as a part of behavioural assessment. Here, questions like *Does the adaptation process lead to a correct result?* and *Is the adaptation process itself correct?* are investigated. Verification and validation methods can be found in a wide range of scientific areas. Although there has been some effort to apply such methods in the context of self-adaptive systems, we could not find any new metrics in the according literature. The existing methods aim to investigate the adaptation process as a whole and give no information on single adaptation events. We refer to [84] as an entry point for the interested reader.

3.6 Summary

As we can see, when it comes to measuring the self-adaptation process itself, the number of metrics that are applicable to a wider range of self-adaptive systems is limited. The metrics for self-adaptation have requirements, such as system-specific definitions, expert knowledge, or information about system states at runtime, that must be met before they can be applied to a particular system.

These limitations justify our own requirements **Req1** to **Reg5** for new metrics. All of the given metrics violate at least one of our requirements making them unfit for our framework if applied in their direct definition. An exception can be constructed for metrics based on system states:

Following **Req2** for the metrics in our framework, we have no information about system states. On the other hand, we have knowledge about the configurations. At this point, we are free to define a configuration as a system state. This allows us to use entropy-based metrics in the framework. The *(Un-)Availability* and the *Working/Adaptaton Time* metrics are the other two metrics that only need knowledge about states. However, they do not only know what state the system is in, but also what the states mean. This information cannot be derived from the configurations.

Using configurations as states has a slight drawback: during runtime, the contents of the set of system states and their probabilities can only be estimated. Nevertheless, with enough collected data, such metrics can be used in our framework.

Thus, only the metrics that use entropy can be used in our framework. Their application works best if the set of possible configurations is known beforehand. Otherwise, methods for the estimation of the entropy have to be used.

For the other approaches, the conclusion is as follows: The measurements of emergence, apart from entropy, require expert knowledge. The homeostasis is not included in the framework because the conversion of a configuration into a suitable string representation needs a definition of *suitable*. The performance metrics can assist our framework in detecting events because changes in performance can be a sign of unusual events in the system. Nevertheless, they are not part of the framework because they do not measure self-adaptation or self-organisation and use other sources of input. The framework of *Miner et al.* wants to measure adaptability and incorporates domain knowledge and has, therefore, a different focus than we have.

Tables 3.1 and 3.2 give a final overview of the existing metrics and approaches, whether they can be used in our framework and the reason for the usage decision.

Metric	**Usable**	**Reason**
Self-adaptation metrics		
Working/Adaptation Time [102]	no	contradicts **Req2**
(Un-)Availability [29]	no	contradicts **Req2**
Decision Benefit [185, 64]	no	contradicts **Req2**, **Req4**
Population Metric [31]	no	contradicts **Req2**, **Req4**
Requirement Performance [13]	no	contradicts **Req1**, **Req4**
Situation Performance [219]	no	contradicts **Req1**, **Req4**
Self-organisation metrics		
Graph-theoretic metrics [21, 240, 103, 193, 226]	no	contradict **Req3**
Interaction metrics [34]	no	contradict **Req3**
System Complexity [21]	no	contradicts **Req3**
Limited Bandwidth Recognition [26]	no	contradicts **Req1**
SO Performance [53]	no	contradicts **Req1**, **Req5**
Oscillation [21]	no	contradicts **Req2**, **Req5**
Entropy [166, 226, 201, 233, 248, 106]	yes	when configurations are considered system states

Table 3.1: Overview of the existing self-adaptation and self-organisation metrics and their usability in the framework and the reason for the classification.

Other metrics and approaches		
Metric	**Usable**	**Reason**
Emergence [205, 174]	no	contradicts **Req2**
Emergence [93]	no	contradicts **Req3**
Emergence [143]	no	contradicts **Req1**
Decentralised Emergence Detection [159]	no	contradicts **Req2**, **Req3**
Homeostasis [66]	no	contradicts **Req1**, **Req2**
Performance metrics [141, 41, 172, 42, 135]	no	contradict **Req1** and have different scope
Miner's framework [142]	no	contradicts **Req1**

Table 3.2: Overview of the other existing metrics and approaches and their usability in the framework and the reason for the classification.

Chapter 4

Measurement Framework

In this chapter, we will describe our framework to evaluate the adaptation process and how it is applied to SASO systems that incorporate our system model as defined in Chapter 2.4. Before we present the framework in Chapter 4.2, we will introduce the underlying core assumption and a few remarks in Chapter 4.1. In Chapter 4.3, we will then introduce our new metrics for the framework and present the existing ones which we will also add. Chapter 4.5 concludes this part of the thesis with remarks on combining the framework with other assessment approaches.

4.1 Assumptions and Remarks

Before we introduce the measurement framework, we will show which role our system model plays and highlight our core assumption. After that, we talk about specific adaptation properties (such as stability) and the distance functions used in the computation of the metrics.

4.1.1 The Role of the System Model

As outlined in the description of the system model, we require an externally readable configuration for every subsystem. This is because we assume a SASO system as a collection of actual technical systems where every subsystem has some kind of configuration and that this configuration determines its current behaviour. Furthermore, we assume that the adaptation process eventually leads to changes in the configuration and that this is the only source of change. Therefore, configuration changes directly reflect the adaptation process.

Based on this idea, it is evident that the analysis of configuration changes can give insight into the adaptation process. Of course, for the analysis of the configuration

changes of a single subsystem, there is no need for new metrics. Its configuration forms a time series, and there is an extensive tool-set for time series analysis with a broad theoretical foundation and a huge corpus of textbooks (see e.g. [221]). However, the focus of this thesis is the general, system-wide behaviour (cf. Chapter 1.2) and the detection of unusual adaptation events. To get the necessary view of the whole system, we cannot reduce our attention to single subsystems. To achieve such an overarching perspective, our metrics make use of the configurations of all subsystems at once.

Figure 4.1 shows this concept. Every subsystem provides an n-dimensional configuration vector. All these configurations are collected for a given point in time.

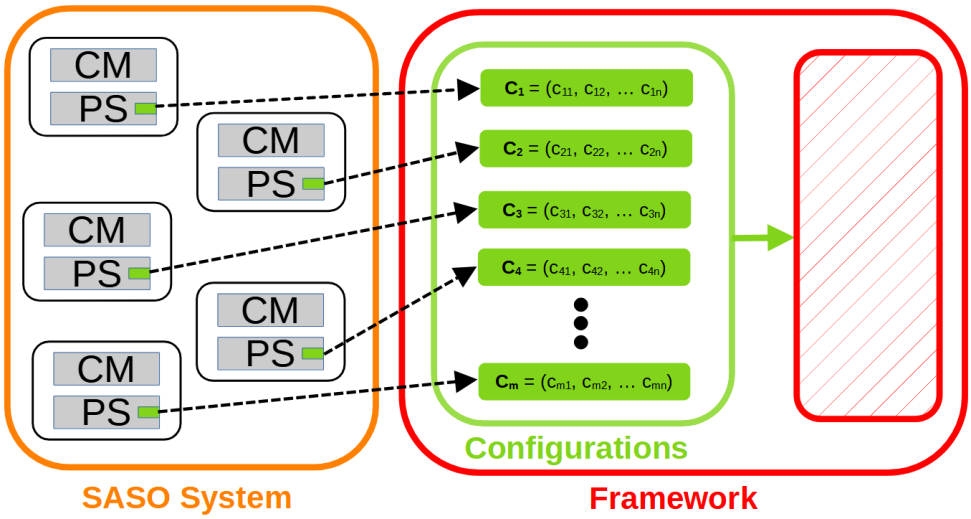


Figure 4.1: Visualisation of the role of the system model as a data source for the framework. At any given time point, every subsystem carries an n-dimensional configuration vector which can be accessed from the outside.

4.1.2 The Core Assumption

The core assumption that all our metrics rely on is that during the normal operation of the system, the configuration changes are small. Although isolated bigger changes in a single subsystem might occur, we assume that widespread bigger changes result from an adaptation process that most likely is not within the definition of *normal operation*. These are the events we want to detect.

4.1.3 Specific Adaptation Properties

We need to note that we are dealing with the raw data of the configurations. Our metrics have no insight into the internals of the adaptation process and decisions. Besides the

detection of adaptation events, they cannot, in general, give information about specific adaptation properties such as stability, safety or appropriateness. Another note is that we use the word metric synonymously with the word measurement, and the axioms for metric mappings in mathematical metric spaces do not hold.

4.1.4 Distance Functions for the Metrics

Some of the following metrics require a notion of distance between two parameter values in their definitions. The explanations around the definitions will make use of the usual Euclidean distance function and terms such as *minimum* and *maximum*. Of course, that applies only to parameters that lie in an ordered Euclidean space. Circular values (e.g. angles of direction), on the other hand, have no natural order and require a different distance function [120]. Furthermore, we will calculate the distances of points on a torus. For that case, a non-euclidean distance function is required, too. We note that the definitions hold for any appropriate distance function.

4.2 The Measurement Framework

We assume that configuration changes reflect adaptation processes and that abnormal adaptations correspond with unusual configuration changes. When lifted to the system-wide level, we need to inspect the configurations of all subsystems at once. Thus, the framework collects the configurations and processes them in several steps. Figure 4.2 shows the procedure:

1. At a given point in time, all configurations are collected from the SASO system.
2. If the communication is accessible, then the current communication messages are recorded.
3. The configurations are stored, and together with historical data for every subsystem, a time series of the configuration is created.
4. If applicable, the *SO Divergence* metric uses the time series of the processed communication data. All other metrics consume the configuration time series of every subsystem. With this input, the metrics calculate a new value.
5. The output from the metrics is stored and then can be accessed as a time series for further usage.

The framework is independent of the SASO system. It requires no knowledge about the processes that lead to the current configurations, no information about the inputs for those processes, and no additional data besides the provided configurations and the communication messages (if available).

The framework and the contained metrics are not intended to provide a quantitative assessment where a particular value x for a metric denotes a specific property or a

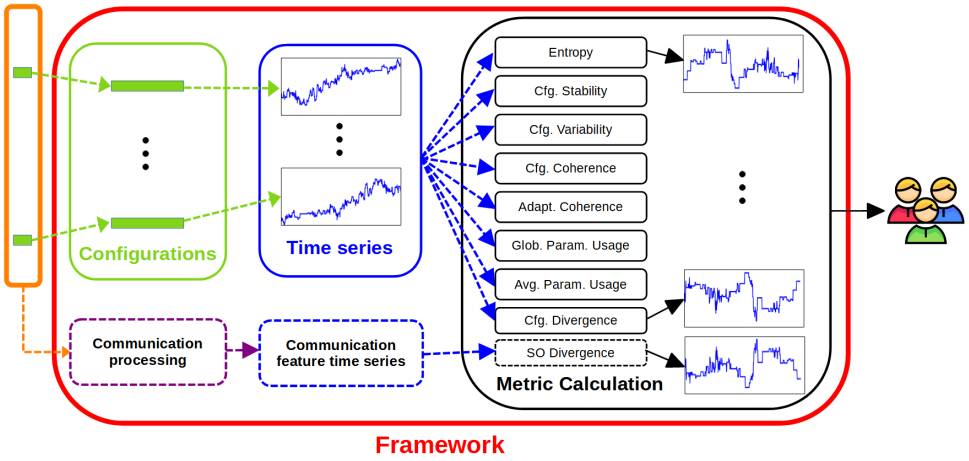


Figure 4.2: The graphical representation of the Measurement Framework and its parts. The SASO system, with its subsystems and observable communication, provides the input for the framework. The framework collects the configurations from the SASO system and monitors and processes available communication. From this input, several time series are created, which act as input to the metrics (black frame, cf. Chapter 4.3). The output of the metric calculation is then provided to the user as time series.

particular class of self-adaptation. Instead, the resulting time series are meant to be evaluated on a qualitative basis where their development over time is analysed.

For example, assume a metric generates a time series that stays within the value range of $[0, 2]$ except for a time window where it stays in $[6, 10]$. In this case, the numbers alone have no meaning regarding the self-adaptation. The only meaningful properties of the time series are the two time points where the interval with the higher values starts and ends and the relative amount of change.

4.3 The Contained Metrics

In this part, we will present the metrics contained in our framework. After motivating the basic ideas behind them, we give an overview of the metrics and then describe them in detail.

4.3.1 Motivation for the Contained Metrics

As we have seen in Chapter 3.6, the existing metrics have requirements (e.g. domain knowledge) that render their out-of-the-box application unfeasible. They have to be adapted to the target system. On the other hand, we want our framework to be applicable at runtime in a wide range of SASO systems and domains and without extensive adjustments. Thus, the contained metrics need to be free from such knowledge restrictions.

For this reason, we postulated the five requirements **Req1** to **Req5** (see Chapter 1.3) that we consider necessary.

Following our core idea that configurations reflect adaptations, all contained metrics use the configuration of the subsystems as their only input. This alone lets them comply with three of our requirements:

- They have no information about internal system states (**Req2**).
- They have no knowledge about the system's structure (**Req3**).
- The outcome of the self-adaptation is irrelevant to them (**Req4**).

Some of the metrics need to be configured with numerical parameters which are system-specific. These parameters can be found by experimentation and experience. Although they are system-specific, we do not see this as expert knowledge because a normal user of the framework is able to identify suitable parameters. Even those metrics that need a predefined range of possible configuration values can be used without knowledge about the actual possible range of values. If the metrics run into a situation where the observed range is exceeded, the metric parameters can be adjusted, and the situation can be marked as exceptional. Therefore, the contained metrics also meet **Req1** (no expert knowledge).

The fifth requirement **Req5** – identification of unusual events with no or an acceptable delay – is open for interpretation. Some metrics react with a delay to events. However, the duration of this delay is acceptable in our eyes, as we will see in the evaluations in Chapter 5.

4.3.2 Overview of the Metrics

As outlined in Chapter 3.6, entropy-based metrics are candidates for the framework if the configurations are viewed as system states. Chapter 4.3.3 introduces a simple metric with this approach, where we will view the configurations as a random variable and then define a simple entropy estimation.

Next, Chapter 4.3.4 presents our first new metric. It continues the idea of seeing the configurations as a random variable and combines it with the notion of active and inactive subsystems. This metric uses a temporal view on the distributions of configurations and a classification of the subsystem. For both of these aspects, we have metrics that follow the other direction: Chapter 4.3.5 investigates the distribution of configuration at a single point in time using a clustering approach and instead of distinguishing several classes, the metrics in Chapter 4.3.6 look at the uniformity and coherence of the subsystems and the system as a whole. Another set of metrics that assesses the whole system and the subsystems is based on the range of used configurations. These are given in Chapter 4.3.7.

Finally, if the SASO system provides communication data, we can cover the self-organisation aspects with the communication analysis from [226]. Here, the communi-

cation messages are transformed into probabilistic distributions that are compared over time. We will include the definition of this metric in Chapter 4.3.8.

4.3.3 Entropy

Since we have no insight into the adaptation process of the subsystems, we are free to see the resulting configurations as a random variable. For such a random variable X and its probabilistic density f we can calculate the differential entropy $h(x)$ as [38]:

$$h(X) := -\int f(x) \log f(x) dx. \tag{4.1}$$

Since we do not know the actual density function f, we have to do an estimation. From the different approaches for entropy estimation (see [15] for an overview), we use the histogram approximation [68]:

For a dataset with N points, a histogram with M bins and n_i points in the i-th bin, let $p_i = n_i/N$ be the fraction of data points in the i-th bin. Then the estimated Entropy H is:

$$H = -\sum_{i=1}^{M} p_i \log (p_i). \tag{4.2}$$

Applied to our SASO systems, the dataset becomes the set of configurations. For the univariate case, this means in detail that if min_N denotes the minimum and max_N the maximum value of all configurations, then with a given number M of bins, the width w of each bin is $w := (min_N - \max_N)/M$. The bounds of the i-th bin are $[min_N + i \cdot w, min_N + (i+1) \cdot w)$. A configuration value c is in the i-th bin if $min_N + i \cdot w \leq c < min_N + (i+1) \cdot w$.

The parameter M depends on the range of possible configuration values and is, therefore, system-dependent. For discrete configuration parameters, M should be the number of possible values. For continuous configurations, a lower bound has to be identified during design time. If M is less than this lower bound, the bin width becomes too big. This can occlude information in the resulting time series. If M is greater than the lower bound, the bins become smaller. This will lead to more bins without content. Those bins do not contribute to the calculation of H. Eventually, this leads to higher values of H because p_i decreases and $\log p_i$ increases. But, as noted above, the actual values of H are not important for our framework – only the development over time.

Sorting N configurations into M bins has a time complexity of $O(N \cdot M)$, followed by M computations of p_i and $p_i log(p_i)$. Therefore, the time complexity of this metric is $O(N \cdot M)$.

4.3.4 Configuration Stability

Our first metric is called *Configuration Stability*. This name is based on the idea that a subsystem which – loosely said – mostly uses configurations in a given time frame that already have been used quite often before expresses a stable behaviour.

As noted above, the decision models used to plan and enact the adaptations are hidden from external observers. Therefore, we can model the configuration changes of single subsystems as a random process with the configuration vector as a random variable. Thus, the observed configurations form a random distribution. For this distribution, an estimated density can be associated, and new configurations can be assigned a probability concerning that density.

Now, the idea is that if new configurations with a high probability are chosen, the system works as expected, and the underlying adaptation is desirable or *normal*. In such a case, we call the configuration state *stable*. If a new configuration has a low probability, we still assume that the system is working as usual and that this adaptation is due to a minor disturbance. Only when configurations with low probability are chosen over a longer course of time, we presume that a significant disturbance or a system failure is at hand.

If it comes down to the implementation, configuration parameters are usually represented as real numbers. Therefore, we will make use of continuous probability distributions and densities. Since a single vector of real numbers has a probability of $P = 0$ in a continuous density, we will look at the probability densities that are created by several vectors. This is achieved by collecting the current configurations at several time points in a given time frame (i.e. a Parzen window [167]).

Figure 4.3 illustrates the idea. Here, the subsystem under investigation has only one configuration parameter a. In the time frame associated with the blue graph, the parameter mostly took values between $a = 0$ and $a = 1$, with more occurrences of values near 0. In the time frame for the red graph, the values, again, were mostly between 0 and 1 and, in this interval, more uniformly distributed than in the other frame. Yet, some more values were near $a = 1$.

By comparing the configurations of a subsystem in one time window with those of a previous window, we can identify abnormal adaptation activity in a single subsystem. Of course, it is possible to choose any time window to compare the current one with. For example, one could use a time window that represents a normal phase of operation. But in fact, the definition of normality can change over time as the result of ongoing changes in the environment and the adaptation of the system. What was normal a long time ago may not be normal now. Therefore, the previous time window is the most reasonable choice.

Before we explain how to compare the configurations (based on their densities), we

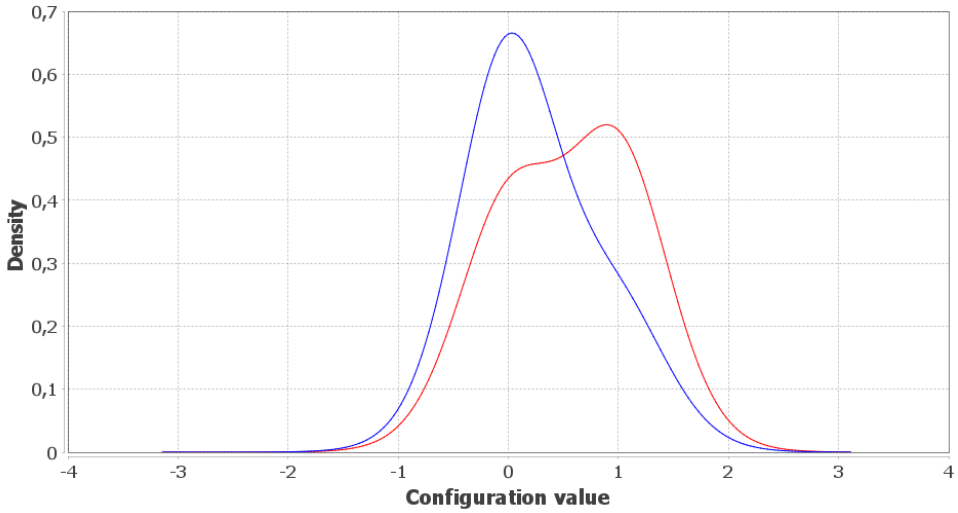


Figure 4.3: Illustration of the density functions for two distributions of a univariate subsystem configuration.

will look at the system level: to identify abnormal changes in a SASO system at a global scale, we take all configuration comparisons into account and then apply a measure based on the *macroscopic measure for detecting abnormal changes in a multi-agent system* as defined by *Kinoshita* [107].

But before we go into the details, we need some preliminary work for the mathematical tools, namely *densities estimators* and *divergence measures*. They help us with the two crucial parts of the computation of the *Configuration Stability*: How to get a probabilistic density from the observed configurations and how to compare the densities.

Density Estimation

To estimate a probability distribution and its associated density function from observed data, several approaches can be used (see [208] for a detailed overview). The obvious approach is a histogram. Here, the data is collected into bins with a given width. This will lead to a stepwise defined, (usually) non-continuous function, and the result depends heavily on the number (and therefore the width) of the bins. Based on this histogram approach, several methods to construct a continuous density function are being used. A widespread method is the *kernel density estimation* [81].

For a set $(x_1, x_2, ..., x_n)$ of observed univariate data, the kernel density estimator function f is defined as

$$f_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{4.3}$$

where K is a *kernel function*, and h is the bandwidth.

A kernel function K is a function that is real-valued, non-negative, integrable, normalised (i.e. $\int K(x)dx = 1$) and symmetric (i.e. $K(-x) = K(x)$). The multivariate case is defined analogously: let H denote a symmetric, positive definite, d-dimensional matrix and K a multivariate kernel function of appropriate dimension. Then the d-dimensional kernel density estimator function f is

$$f_H(x) = \frac{1}{n}\sum_{i=1}^{n} K_H\left(x - x_i\right), \tag{4.4}$$

where the x_i are d-dimensional data points and $K_H(x) = |H|^{-1/2}K(H^{-1/2}x)$.

For more details on kernel density estimation and especially the bandwidth parameter see [81].

For the evaluation in Chapter 5, we will use this method of density estimation for two reasons: first, there are methods for computing a reasonable bandwidth parameter h, which the used statistical computation libraries already implement. The second reason is the easy way to plug in the kernel functions. We will use two different kernels: the Gaussian kernel $K(x) = \frac{1}{\sqrt{2\pi}}e^{-0.5x^2}$ for non-circular parameters and a von-Mises kernel $K(x) = a * e^{cos(x\kappa)}$ for circular values such as angles. For the specialities of circular kernel functions and the parameters in the von-Mises kernel see [164].

At this point, we need to note that the multivariate case needs some caution. The higher the dimension is, the more sparse the data points are in the increasing volume of the parameter space. The curse of dimensionality [18] is striking. Therefore, we suggest carefully selecting the configuration vector entries that should be monitored together and excluding the less important ones. If necessary, every entry in the vector can be monitored on its own and separate time series for each entry can be investigated.

Divergence Measures

In statistics, divergence is a concept of distance and is used to quantify the dissimilarity of probability distributions. There are several established divergence measures used in statistical applications. Due to its close relation to entropy and its applications to continuous time series, we will stick to the *Kullback-Leiber divergence* KL: for distributions P and Q of a continuous random variable, it is defined as

$$KL(P,Q) = \int_{-\infty}^{\infty} P(x)log\left(\frac{P(x)}{Q(x)}\right)dx. \tag{4.5}$$

For our application, we are interested in a quantification of the distance between P and Q regardless of the order of presence. Therefore, we will use the symmetric version KL_2

as

$$\begin{aligned} KL_2(P,Q) &= \frac{1}{2}\left(KL(P,Q) + KL(Q,P)\right) \\ &= \frac{1}{2}\int_{-\infty}^{\infty} P(x) log\left(\frac{P(x)}{Q(x)}\right) dx \\ &+ \frac{1}{2}\int_{-\infty}^{\infty} Q(x) log\left(\frac{Q(x)}{P(x)}\right) dx \\ &= \frac{1}{2}\int_{-\infty}^{\infty} P(x) log\left(\frac{P(x)}{Q(x)}\right) + Q(x) log\left(\frac{Q(x)}{P(x)}\right) dx. \end{aligned} \tag{4.6}$$

For a deeper dive into divergences, we refer to [22].

The Definition of the Kinoshita Measures

Kinoshita proposes a measure that he uses as an indicator for unusual activity changes in distributed multi-agent systems. Kinoshita defines two values: the activity factor at a given time and the variance of fluctuation of the activity factor. The activity factor is based on the classification of each agent as either *active* or *inactive*. The definition of *active* is specific to the investigated system.

Let N denote the total number of agents in the multi-agent system, and let n_t be the number of active agents at a given time t. The activity factor z_t is defined as

$$z_t := \frac{2 \cdot n_t - N + 1}{2 \cdot N}. \tag{4.7}$$

For a given window size M, the fluctuation ξ_t of the activity factor at time t is calculated as

$$\xi_t := z_t - \frac{1}{M} \cdot \sum_{i=0}^{M-1} z_{t-i}, \tag{4.8}$$

and finally, we calculate the statistical variance ν_t of the fluctuation:

$$\nu_t := \frac{1}{M} \sum_{i=0}^{M-1} \xi_{t-i}^2 - \left(\frac{1}{M} \sum_{i=0}^{M-1} \xi_{t-i}\right)^2. \tag{4.9}$$

Kinoshita's publication is based on the working hypothesis that unusual peaks in the time series of ν_t indicate abnormal changes in the underlying distribution of active and inactive subsystems. He backs his hypothesis with theoretical reasons and an experimental evaluation. In Chapter 5, we will see that this idea is reasonable in our cases.

From the Kinoshita Measures to the Configuration Stability Metric

To apply the Kinoshita measure, we need to define what active and inactive agents are. Obviously, the subsystems of a SASO system are the agents in Kinoshita's measure. We define a subsystem as active if and only if its configuration is unstable. That means that for the latest configurations, values were chosen that follow a distribution that significantly differs from the previous one. This difference is expressed as the result of the symmetric Kullback-Leibler divergence function (see Chapter 4.3.4). So, if the divergence of the density of the latest configurations and the density of previous configurations is greater than a certain threshold, then we call that configuration unstable, and the subsystem is labelled as active.

Let $c_{a,t} \in \mathbb{R}^n$ be the n-dimensional configuration vector of the subsystem a at time t. We model $c_{a,t}$ as a random variable. Let $M > 1$ be the window size, and $L > 0$ be the delay parameter, i.e. the offset between the two time windows. Using an appropriate kernel for the density estimation (see Chapter 4.3.4), we define two probability densities $DC_{a,t}$ and $DP_{a,t}$ based on parts of the time series of $c_{a,t}$:

$$DC_{a,t} := density(c_{a,t-m}, c_{a,t-m+1}, ..., c_{a,t}), \tag{4.10}$$

$$DP_{a,t} := density(c_{a,t-l-m}, c_{a,t-l-m+1}, ..., c_{a,t-l}). \tag{4.11}$$

DC is the density of the current window, and DP is the density of a previous window in the time series. Now, we apply the symmetric Kullback-Leibler divergence measure to these two densities and get a new time series

$$d_{a,t} := KL_2\left(DC_{a,t}, DP_{a,t}\right). \tag{4.12}$$

Finally, we define a subsystem a to be active at time t if $d_{a,t} > \varepsilon$ for a threshold value ε. Otherwise, the subsystem is called inactive. With this distinction between active and inactive, we call *Kinoshita's* ν_t the *Configuration Stability* at time t. For a consistent notation, we will denote *Configuration Stability* with c_s.

Time Complexity

The first part of the computation is the classification of active and inactive subsystems. The kernel function has to be computed based on $2M$ points, M points for each of the two time windows. For the computation of the integral, the kernel function has to be evaluated at p points, where p is a constant that defines the accuracy of the result. The more points are evaluated, the more accurate the result becomes.

The core of the second part is the computation of the statistical variance of M values, which is $O(M^2)$. The overall complexity is then $O(M^2) + O((M + p) \cdot N)$.

4.3.5 Configuration Variability

The idea behind this metric is again that in a stable environment, new configurations do not differ much from the previous ones, and only minor adaptations take place. Therefore, the set of all configurations at a given time can be divided into several clusters such that the diameter of every cluster is relatively small, and for all points in the configuration space, the average distances to their respective centroid will not change much. So, if we monitor an unusual change in the time series of the average distances, we either have a change in the number of clusters or the diameter of at least one cluster changed notably. That means at least one subsystem switched to a configuration that differs significantly from its previous one. This serves as an indicator of unusual adaptation processes of which the user should be informed.

Definition of the Configuration Variability

To calculate the *Configuration Variability*, we cannot fix a number of clusters in advance for two reasons: 1) Clustering algorithms can produce different results if a single data point moves slightly, and 2) we want to be able to apply this measure to a variety of system architectures and scenarios which all come with different layouts of typical configuration distributions. Therefore, we will take several values for the cluster count into account. First, we calculate the average centroid distance for a given cluster count:

Let k be the number of clusters, X_i a single cluster ($1 \leq i \leq k$) and C_i the centroid of X_i. First, compute the distances for every data point p in X_i to C_i, take their sum and then divide by the cluster size to obtain the average distance d_{X_i} in the cluster:

$$d_{X_i} := \frac{\sum_{p \in X_i} \|p - C_i\|}{|X_i|}. \tag{4.13}$$

Having the average distances, we now compute the average s_k over all clusters:

$$s_k := \frac{\sum_{i=1}^{k} d_{X_i}}{k}. \tag{4.14}$$

Next, we compute s_k for several values of k. The higher k is, the fewer data points are in each cluster, and the cluster diameters shrink. When k reaches the number of subsystems, all clusters will contain at most one single point. With increasing k, the value of s_k will decrease towards 0. Hence, higher values will give no more information. On the other hand, low values of k might not capture the real cluster layout at a given time. There are methods to determine the best k for optimal clustering (see [137]), but that value is usually not constant over time, even for a single system. Therefore, we define the upper bound for k as the square root of the number of subsystems, which is a

trade-off between lower and higher values. If necessary, we round up to the next integer:

$$k_{max} := ceil\left(\sqrt{|S|}\right). \tag{4.15}$$

As a subsequent step, we define the *Configuration Variability* c_v as the average of s_k over the clusterings for $k = 1...k_{max}$

$$c_v := \sum_{k=1}^{k_{max}} s_k/k_{max}. \tag{4.16}$$

That means we divide the data points into clusters for several cluster counts and then take the average distance over all these counts. This approach is justified because we are not interested in the actual cluster layouts or in the fact that they are optimal. Multiplying every data point with a scaling factor will change the resulting value of c_v accordingly. This is no restriction because we are only interested in the development of the resulting time series. The experiments in Chapter 5.2 show that the choice of the actual clustering algorithm has no qualitative impact on this development of the time series. Low values for this metric result from the configurations being close to their respective centroids. That means that they do not vary much within their cluster. This also explains the name of this metric.

Extension to Time Windows

The metric is calculated over a set of configuration values. This set is created from the values at a single point in time. Likewise, we could collect the values over a period of time. Using larger time windows will increase the number of the considered values, and the impact of outliers will be reduced. Whether this approach is beneficial depends on the actual SASO system under investigation.

Time Complexity

For the k-means cluster algorithm, the time complexity is $O(Nkl)$ for N data points, k clusters and l iterations of the algorithm [137]. Since we repeat the calculations for $k = 1...k_{max}$, we do a clustering $(N + \sqrt{N})/2$ times in total. Thus, the complexity of the *Configuration Variability* with k-means is $O(l(N^2 + N\sqrt{N})) = O(lN^2)$.

4.3.6 Coherence

As mentioned above, during the normal operation of the system, the configuration changes are small. The following Coherence metrics express this assumption with a mathematical distance function on the configuration spaces. Therefore, if the adaptation

process of a group A of subsystems leads to configurations that differ strongly from those of other subsystems, then we can assume that the members of A experience noteworthy conditions (such as strong changes in the environment, internal or external disturbances).

Since we aim to measure these changes on a global scale, we define two measures: one that takes all configurations at a given time into account, as described in Chapter 4.3.6, and one that only looks at those configurations that have changed (see Chapter 4.3.6). If the configuration changes of the subsystems are close to each other, we call these changes *coherent*. If they differ strongly, we call them *incoherent*.

If no adaptation occurs for a given time point, then no variance can be computed. For the time series, we find it beneficial to repeat the last computed value in such a case.

Definition of the Configuration Coherence

Let S be the set of all subsystems and $x_s \in \mathbb{R}^n$ the configuration vector of a subsystem $s \in S$ at a given time point t and $\hat{x}$ their statistical average. Let $v_S := \sigma_S^2 = |S|^{-1} \cdot \sum_S \|x_s - \hat{x}\|^2$ be the statistical variance of all configuration vectors.

We then define the **Configuration Coherence** coh_c of the system as

$$coh_c := \frac{1}{1 + v_S}. \tag{4.17}$$

The value of coh_c is in $(0; 1]$ and is 1 if and only if all configuration vectors are identical. Therefore, a higher value means a higher coherence. Since we are looking at all subsystems, this value can be used to define a *(configurational) coherence* of the whole system.

Definition of the Adaptation Coherence

We call the second metric **Adaptation Coherence**. In this case, we only take the configuration vectors of subsystems that have changed their configuration since the last observation. We denote the set of these subsystems as $A \subseteq S$. The rest of the definition is as above:

$$coh_a := \frac{1}{1 + v_A}. \tag{4.18}$$

Extension to Time Windows

The coherence metrics are calculated over a set of configuration values. This set is created from the values at a single point in time. Likewise, we could collect the values over a period of time. Using larger time windows will increase the number of the considered values, and that will lead to a smoother time series, as the impact of outliers will be

reduced. Whether a smoothing is beneficial depends on the actual SASO system under investigation.

Time Complexity

The selection of the subsystems for the *Adaptation Coherence* is $O(N)$. The computation of the statistical variance of N values is $O(N^2)$, which is the resulting complexity for both metrics.

4.3.7 Parameter Usage

For these metrics, we will examine the intervals in which the configuration values lie. For this, we require all subsystems to be of the same architecture, i.e. the control variables and parameter ranges are identical for all subsystems. Then, we can compare the parameter intervals. For the *Global Parameter Usage*, we will investigate the parameter interval of all subsystems together, i.e. the smallest and the biggest values taken over the whole system. The parameter intervals of each subsystem on its own are treated by the *Average Parameter Usage*.

Definition of the Global Parameter Usage

First, we fix a point t in time and a window size M. We then collect all configuration vectors from all subsystems in the time frame F from $t - M$ to t. For each individual control variable j, we then extract the minimum $min_j(t)$ and maximum value $max_j(t)$ that occurs in F over all subsystems. We now compute the effective parameter range $V_{j,t} := d(min_j(t), max_j(t))$ of the current configuration space (using an appropriate distance function d). $V_j := d(min_j, max_j)$ will denote the maximum parameter range that the control variable j can take without restriction to a time frame. Finally, we define the *Global Parameter Usage* $U_{j,g}$ as the fraction of the current range in the total range:

$$U_{j,g}(t) := \frac{V_{j,t}}{V_j}. \tag{4.19}$$

In geometric terms, we compute the volume of the parameter space (current and total) for each variable (hence the letter V). Hereby, we calculate the value for every variable individually. Of course, we could calculate the volume for more than one variable at once and receive a hyper-cube, but usually, not all variables change in a given time frame. That would lead to an effective volume of 0, although some variables did change.

Definition of the Average Parameter Usage

For this metric, we calculate the parameter usage in the frame F for every subsystem individually: for the subsystem a_i and every control variable c_j, we get the minimum and maximum value that c_j takes in the time frame defined by t. Those define the parameter range that a_i used in F:

$$V_{t,j,a_i} := d_j(max(c_{j,a_i}), min(c_{j,a_i})). \tag{4.20}$$

The *Average Parameter Usage* $U_{j,a}$ is then defined by the average of all these ranges divided by the total range and the number of subsystems:

$$U_{j,a}(t) := \frac{\sum_{a_i \in A}(V_{t,j,a_i})}{|A| \cdot V_j}. \tag{4.21}$$

Evaluating the time series of the parameter spectrum usage can give pointers to situations with "abnormal" adaptations, which we will see in Chapter 5.

A Simple Example for the Parameter Usage

Let us assume a SASO system with 4 agents. These agents have a single configuration parameter p that can range from $p = 0$ to $p = 10$. At a certain time window in time, we might have seen agent 1 take values for p in the interval $[4, 6]$, agent 2 uses $[3, 6]$, agent 3 is in $[4, 8]$ and agent 4 uses $[3, 5]$. Therefore, the whole system has an actual parameter range from $p = 3$ to $p = 8$. Figure 4.4 illustrates the intervals.

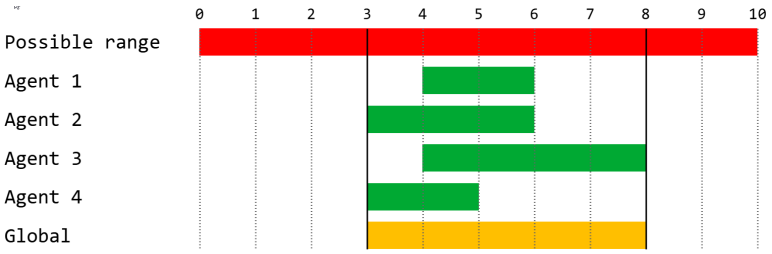


Figure 4.4: Illustration of the example for the calculation of the Parameter Usage metrics

So the Global Parameter Usage in this example is

$$U_g = \frac{\mathbf{5}}{\mathbf{10}} = 0.5 \tag{4.22}$$

and the Average Parameter Usage gives

$$U_a = \frac{\mathbf{2 + 3 + 4 + 2}}{\mathbf{4 \cdot 10}} = 0.275. \tag{4.23}$$

The Influence of the Window Size

The window size M is the only parameter required for the evaluation. Clearly, for the average parameter usage, M must be greater than 0. Otherwise, all parameter ranges would contain only one point, which leads to a usage value of 0. Higher values of M result in smoother graphs for the parameter usages but might also mask short-term effects. Lower values generate noisier graphs and might lead to false-positive detection. In our experiments, we found that $M = 5$ is a suitable trade-off for the given scenarios. Nevertheless, an acceptable value for M depends on the evaluated system.

Time Complexity

For the *Global Parameter Usage*, we evaluate N subsystems at M time points. This gives a complexity of $O(NM)$. The same complexity applies to the calculation of the average in the *Average Parameter Usage*.

4.3.8 Distribution Divergence Metrics

As outlined in Chapter 4.3.4, we can associate sets of observed data with probabilistic distributions and calculate the KL_2 divergence. This approach leads us to two very similar metrics.

Divergence of Self-Organisation Messages

The first metric was introduced by *Tomforde et al.* [226] with the goal of measuring self-organisation at runtime. The metric assumes that the self-organisation process in a SASO system is managed using special messages that the subsystems exchange. If there are no such messages present, then no self-organisation takes place. Every message has diverse attributes such as origin, destination or a time stamp. Since the subsystems act as black-boxes, these attributes can be seen as random variables. Therefore, we can monitor the self-organisation messages in the system over a period of time and then assign a probabilistic density to the distribution of the observed attributes. This density then can be compared to the density of a previous observation period. The authors state that if there is no difference between the two densities (i.e. the distributions of the attributes are identical), then the system is maintaining its current organisational structure. That means that no self-organisation takes place. Only if the distributions show unexpected or unpredictable changes, the system is currently self-organising.

Let $M > 1$ be a window size, and $L > 0$ be a delay parameter. Let C_P be the set of self-organisation messages in the time frame $[t-M-L, t-L]$ and C_Q the set of messages in the time frame $[t-M, t]$. Using the methods from Chapter 4.3.4, we obtain a density

Q for the set C_Q and P for C_P. With the notations from Chapter 4.3.4, we define the *Self-Organisation Divergence* $SOD(t)$ as

$$SOD(t) := KL_2(P, Q). \tag{4.24}$$

Divergence of Configurations

The second metric uses the configurations instead of processed messages. This is comparable to the approach in the *Configuration Stability* metric, where we used the result of the KL_2 divergence of the single subsystems. Now, we use the configurations of all subsystems:

Let M and L be as above. This time, C_P is the set of configurations of all subsystems combined in the time frame $[t - M - L, t - M]$, and C_Q is the set of configurations in the time frame $[t - M, t]$. Again, we form the according densities P and Q, and then we calculate the *Configuration Divergence* $c_d(t)$ as

$$c_d(t) := KL_2(P, Q). \tag{4.25}$$

Time Complexity

As we have seen in the Configuration Stability, the time complexity for the KL_2 calculation is $O((M+p) \cdot N)$ for N subsystems, a window size of M and the precision parameter p.

4.3.9 Finding the Right Parameters

Finding suitable values for the parameters of the metrics is crucial for the significance of the results. For example, a low value for ε in the *Configuration Stability* will mark configuration changes as unstable which, at a closer look, are still acceptable as stable. On the other hand, too high values for M and L in the *Configuration Stability* would mask short-term occurrences of high amplitude changes.

Our evaluation in Chapter 5 will show that good choices for these parameters highly depend on the actual SASO system. Thus, the configuration of these parameters is application-specific and should be customised at runtime using hyper-parameter-tuning. However, in this thesis, we aim at a 'good-enough' reference configuration of parameters.

A first approach to finding such acceptable values is to create a simulation of the SASO system and gather the configuration time series of the simulated subsystems. In the next step, we change these data samples for some of the subsystems such that high amplitude changes occur at controlled time points for a few steps. Then, we create an optimisation problem: find the best values for the parameters such that the controlled changes create unique peaks in the metric output.

Experiments show that the best values vary depending on where we set the artificial changes. However, they vary only in a very limited range. Therefore, a reasonable next step is to run this optimisation several times with different points for the changes. This gives a distribution for parameters, and the values that occur most often are reasonable for the actual application.

Of course, this method only works if it is possible to create such artificial changes and the metrics show a sufficient impact. Otherwise, expert knowledge and system-specific approaches are needed. In fact, in some cases, it might happen that normal and abnormal adaptation behaviour cannot be distinguished by a metric, regardless of the used parameters. If that occurs, this metric is simply not applicable in the given system.

4.4 Comparison of the Metrics

Our metrics all have different requirements, and Chapter 5 shows that not all of them are useful in all scenarios. We will now give a short summary of the requirements and restrictions. We will compare the tuning parameters, the applicability to multivariate configurations, the use of time windows, the applicability to heterogeneous systems and the lag of event detection.

4.4.1 Tuning Parameters

The tuning parameters for the metrics are crucial for their effectiveness. The more parameters there are, the harder it gets to find suitable values for them.

Both *Divergence Metrics* require two parameters, M and L, for the window size and delay. The *Configuration Stability* even requires the threshold ε as a third parameter. For the *Parameter Usage*, the only parameter is the window size. Our *Entropy* metric needs one parameter: the number of bins for the histogram. The *Configuration Coherence* and the *Configuration Variability* can be computed without any additional parameters, although they can be extended to a windowed version. In that case, we need one parameter (again, the window size).

4.4.2 Applicability to Multivariate Configurations

All metrics can be applied to multi-dimensional configuration vectors. Although this is possible even for the *Parameter Usage* metrics, it is usually not reasonable, as explained in Chapter 4.3.7. Nevertheless, it might be beneficial to not apply the metrics to the entire configuration vector. Excluding specific configuration entries can give a better insight into the system dynamics. This, of course, depends on the actual application.

4.4.3 Time Windows

Metrics that use time windows need specific attention to the length of the windows. Too long windows can draw the curtain over noteworthy events. Too short lengths can create unwanted noise in the generated time series. The *Configuration Stability* and the two *Divergence Metrics* are specifically prone to this since they use two distinct time windows. The time window for the *Parameter Usage* metrics is mandatory because otherwise, the minimum and maximum values would be identical. For the *Coherence* metrics, the *Configuration Variability* and the *Entropy* metric, only the current set of configurations is needed. Nevertheless, these metrics do calculations on a set of vectors, and this set is not limited in size. It is possible to collect configurations during a time window and put them in a single set. Whether this approach gives more information than the no-window setting has to be investigated for each specific application of the framework.

Another point of attention is that any time window approach can lead to a delayed detection of events because it can take some time steps before an outlier creates sufficient impact on the value of the metric. Therefore, the window sizes have to be chosen with care. This, again, depends on the actual application with its specific setup of the SASO system and the tuning parameters of the metrics.

4.4.4 Summary

Table 4.1 summarises the attributes of our metrics. The column *Parameters* lists the number of required tuning parameters. For the windowed versions of the *Configuration Coherence*, the *Adaptation Coherence*, and the *Configuration Variability*, we would need to put a 1 here. Under *Multivariate*, the applicability to multivariate configurations is listed. The entries in *Windowed* show whether the metric uses time windows. The column *System* tells us if the metric can be used in heterogeneous systems with mixed subsystems or only in uniform systems where all subsystems are identical. Finally, *Complexity* shows the computational time complexity for N subsystems and the specific parameters of the metrics.

4.5 Concluding Remarks

Let us close this chapter with a look at the way from measurement to assessment, a note on runtime assessment in general, and a remark on heterogeneous systems.

Metric	Parameters	Multivariate	Windowed	System	Complexity
Entropy	1	yes	possible	uniform	$O(N \cdot M)$
Cfg. Stability	3	yes	yes	mixed	$O(M^2) + O((M+p) \cdot N)$
Cfg. Variability	0	yes	possible	uniform	$O(lN^2)$
Cfg. Coherence	0	yes	possible	uniform	$O(N^2)$
Adapt. Coherence	0	yes	possible	uniform	$O(N^2)$
Glob. Par. Usage	1	not advised	yes	uniform	$O(NM)$
Avg. Par. Usage	1	not advised	yes	uniform	$O(NM)$
Cfg. Divergence	2	yes	yes	uniform	$O(N \cdot M)$
SO Divergence	2	yes	yes	mixed	$O(N \cdot M)$

Table 4.1: Summary of the attributes of our metrics.

4.5.1 From Measurement to Assessment

Our measurement framework provides several time series that aim to assess the self-adaptation and self-organisation behaviour of a SASO system. Of course, the time series on their own are only sequences of numbers. For the actual interpretation of the system behaviour, further steps are needed. Since the goal of the assessment is to give indicators of unusual behaviour at runtime, peak detection and comparison with historical data are the most promising approaches because they can react quickly to changes in the time series and are easy to apply. For some systems, even the naked eye of an experienced system user can identify changes in the time series. For the evaluation of the metrics in Chapter 5, we will use the comparison with historical time series and the naked eye. For the application of the framework presented in Chapter 6, we will implement the more accurate approach of peak detection.

Naturally, other approaches from the domain of time series analysis, such as spectral analysis or auto-correlation analysis, can be used. Their usefulness has to be evaluated for the specific application. The investigation of these techniques is beyond the scope of this thesis.

4.5.2 General Runtime Assessment of SASO Systems

Self-adaptation and self-organisation are the defining properties of SASO systems, but for a complete runtime assessment of such a system, other aspects should also be investigated. The monitoring of system performance and resource consumption, the assessment of goal attainment or task efficiency, and the measuring of failure tolerance and failure rates can accompany our framework to answer the questions like *"Is the system running as intended*

regarding the goals?" and *"Is the observed behaviour, regarding performance, error rates or adaptations, normal?"*

4.5.3 Application to Heterogeneous Systems

As we have seen above, most of our metrics are only applicable to homogeneous systems. Only the *Configuration Stability* and the *SO Divergence* can be used on subsystems with a differing layout of the configuration vectors. In general, using only these two metrics is not sufficient for an effective adaptation assessment.

There are two cases where we can bypass this limitation. The first case is when every subsystem contains an identical base component that is extended with additional capabilities and additional configuration entries. Here, we can restrict the analysis to the configuration of the base component and ignore the additional configuration entries that differ for the extensions. The second case is when the SASO system comprises subsystems that cluster identical smaller subsystems. An example of such systems are those where a single control mechanism steers several identical productive systems. In this case, we can simply take each of the actual productive systems and read out their configuration. This is feasible because we are not interested in any control structures of the subsystems.

A third imaginable scenario where we can use our framework in heterogeneous systems is a system that consists of a reasonably small amount of classes of subsystems, and the number in each class is sufficiently high. In this case, we can restrict the application of the framework to only one class of subsystems at a time. This approach does not guarantee a helpful insight into the adaptation process of the whole system but is at least a first step.

Chapter 5

Evaluation

After having established the framework and the contained metrics, we will now apply the framework to evaluate how the metrics behave in different scenarios. Most of the scenarios include a disturbance. We will investigate how far the metrics can act as indicators for these events. For this, Chapter 5.1 introduces the scenarios in detail, performs a taxonomical placement, and a classification with respect to several qualitative properties. In Chapter 5.2, we then show the results and a short summary for each scenario. Chapter 5.3 concludes this part with an overarching summary of the results.

All scenarios are implemented in the Java-based MASON [132] simulation environment and run in discrete time steps. The framework and the metrics are implemented in Java. The source code is available at [263]. For the clustering algorithm in the *Configuration Variability*, we used the implementation of the X-Means [169] clustering algorithm provided in the JSAT Java package [178]. We will provide a comparison with other clustering algorithms in Chapter 5.2.1. We will see that the actual algorithm can be chosen freely since the results show only minimal differences. The only restriction is the fact that the implemented algorithm must be able to be configured to output a given number of clusters.

5.1 Scenarios

The evaluation scenarios in which we will apply and analyse our metrics are

- a flocking scenario,
- a scenario with a smart camera network,
- three different road network scenarios with smart intersections,
- a scenario with real-world motion data from ships,
- and *Conways's* Game of Life.

After describing them in detail, we will apply the taxonomy and explain why we chose them.

5.1.1 Flocking

The flocking simulation consists of 50 birds with random starting points and random initial orientations on a toroidal plane of size 150 by 150 units. All birds follow the usual rules of flocking [187]: for every bird and each rule, a direction vector is computed, weighted with a factor and then added together. The result is normalised and then added to the current direction vector of the bird.

Let t be a fixed time step in the simulation, a be a bird in the flock and $N_a := \{x : x \text{ is a bird and } d(x,a) <= 10\}$ the set of all neighbours of a, i.e. all birds within 10 units distance. Let $v(a)$ be the current direction vector of bird a and $p(a)$ its current position vector. The rules are then in particular [187]:

- **Alignment** - A bird will align its direction with the average direction of its neighbours:

$$v_{avg} := \frac{\sum_{x \in N_a} v(x)}{|N_a|} \tag{5.1}$$

$$align(b) = \omega_a \cdot (v_{avg} - v(a)), \tag{5.2}$$

  where ω_a is a tuning parameter. In our scenario, we choose $\omega_a = 1$.
- **Cohesion** - A bird will steer towards the centre of all neighbouring birds:

$$p_{avg} := \frac{\sum_{x \in N_a} p(x)}{|N_a|} \tag{5.3}$$

$$cohesion(a) = \omega_b \cdot (p_{avg} - p(a)), \tag{5.4}$$

  where ω_b is again a tuning parameter. In our scenario, we choose $\omega_b = 0.1$.
- **Avoidance** - A bird will steer away from neighbours that are too close:

$$h := \sum_{x \in N_a} (p(a) - p(x)) \tag{5.5}$$

$$avoidance(a) = \omega_c \cdot h. \tag{5.6}$$

  The tuning parameter ω_c is $\frac{400}{3}$ for our simulation.

With the resulting sum $s(a) := align(a) + cohesion(a) + avoidance(a)$ and its length $\|s(a)\|$, the position $p_{t+1}(a)$ of the bird a is then updated for the next simulation step $t+1$ as

$$p_{t+1}(a) = p_t(a) + \frac{s(a)}{\|s(a)\|}. \tag{5.7}$$

Our metrics aim to identify unusual adaptation behaviour. Since every adaptation in this scenario is the result of the flocking rules, every adaptation is considered as usual. Therefore, we introduce an event of unusual adaptation as follows: at time point $t = 500$, one bird a_0 is being shot at. All birds within a distance of 50 units fly diametrically away from a_0 for three time steps and ignore the rules during this time. After four steps, they follow their usual behaviour again. Figure 5.1 shows the simulation at $t = 500$ with the result of the shot.

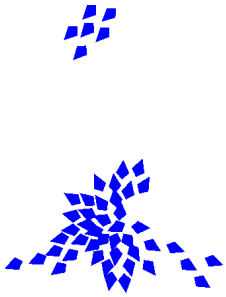


Figure 5.1: The flocking simulation at $t = 500$. The shot occurred near the centre of the lower cluster.

Additionally, we will investigate two more configurations of the disturbance. After the first setting with an influence radius of 50 units and three time steps of disturbance, we will use a smaller disturbance with a 25-unit radius and only one time step length and then an even smaller configuration with a radius of 10 units and one time step length. Comparing the three settings will give us insight into how the different strengths of the disturbance influence the metrics.

For the configuration parameter that is investigated by the metrics, we take the angle of the direction of the birds. The simulation represents this angle as radians with values from $-\pi$ to π.

5.1.2 Smart Camera Network

This scenario is inspired by a museum at night. Figure 5.2 shows the setup. Ten cameras are installed on one side of two hallways and monitor the opposite wall, where the valuable paintings hang. The cameras communicate with each other and share their current viewing angle with their neighbours. Each camera then turns around such that the overlap of their viewports is minimised. Furthermore, each camera will track a person that is inside the viewport. For this, it adjusts its angle such that the tracked person is always in the centre of the viewport. The tracking has precedence over the minimisation of overlaps. If there is no overlap and nothing to track, the cameras do not turn. Of course, the tracked persons are part of the environment. Every 50 time steps, the tracked

person leaves the viewport of the current camera and enters the next one. At these points, the new camera will perform a sharp turn to center the tracked person in the viewport. The viewports of the two cameras on the left partially observe the connecting hallway in the left part of the picture.

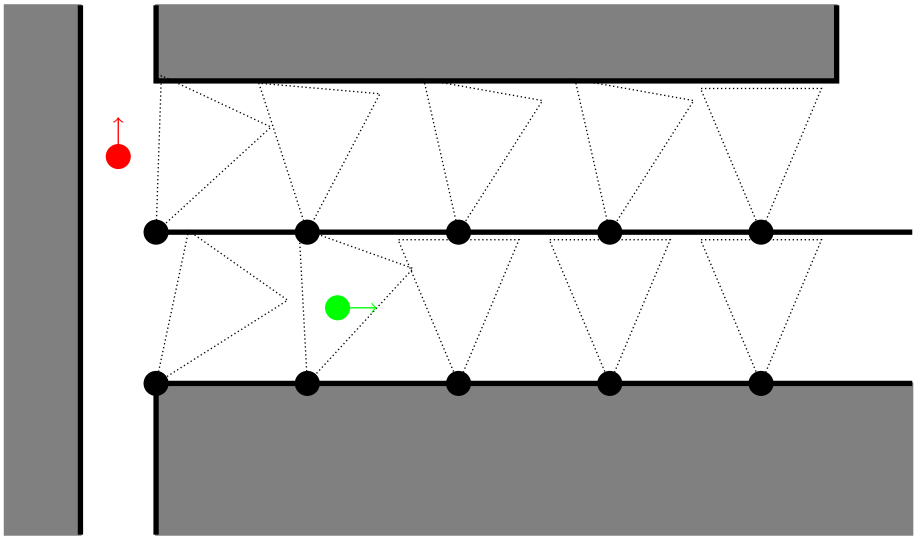


Figure 5.2: The hacked camera simulation at $t = 112$. The hallways are coloured white, the tracked person is green, and the intruder is red. The black dots depict the cameras and the triangles their viewports.

For this scenario, we created two simulations. In the basic simulation, a guard starts in the lower left corner and walks down the lower hallway. The cameras will turn according to their rules. The cameras in the upper hallway do nothing.

In the second simulation, an intruder wants to cross the left hallway from bottom to top. He starts at $t = 40$ when the guard has entered the lower hallway. Since the path of the intruder crosses the viewport of the upper left camera, the intruder hacks that camera and forces it to turn away. Furthermore, the intruder has insider information and knows that the camera network itself is monitored by software that detects anomalies. That software uses our *Configuration Stability* metric to do so. Therefore, the intruder not only turns the upper left camera away from his path but also turns the four other cameras in the upper hallway a little such that the resulting angles will cancel out each other in the sense of the stability metric. They are also small enough to not trigger the overlap minimisation process. This manipulation takes place every time step from $t = 110$ until $t = 120$.

The configuration parameter which we will evaluate in this case is the angle of the viewport for each camera, again, as radians with values from $-\pi$ to π.

5.1.3 Artificial Road Network - Blocked Road

The third scenario is inspired by the Organic Traffic Control (OTC) system that self-adapts the green light duration of traffic lights [176]. Here, we use an abstract traffic simulation where the SASO system consists of seven interconnected intersection controllers. Figure 5.3 shows the layout of the street network.

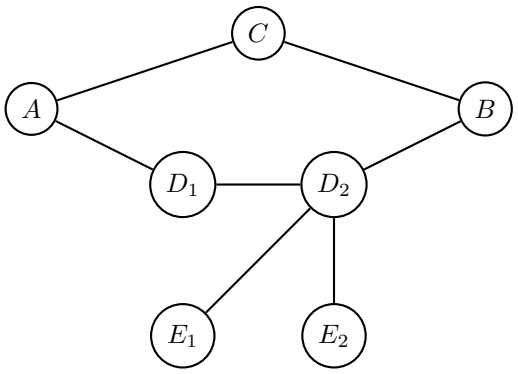


Figure 5.3: The street network for the roadblock scenario. The circles are intersections, and the lines represent the connecting streets. Every street has two lanes, one for each direction.

Each intersection tries to minimise the waiting time for all cars at all incoming lanes by optimising the red light times for each lane. In this abstract simulation, the waiting time for a car A is the number of cars that have passed the intersection since A has arrived. In each simulation step, the intersection lets a certain number of cars in each incoming lane pass the intersection in the desired direction. Since the cars can turn around and drive back, the intersections with only two connected streets need red lights. The intersections know the current waiting time of the cars, and the optimisation aims to find the optimal number of cars to let pass in each lane to minimise the waiting time of the remaining cars that could not pass in this simulation step. The configuration parameters that are investigated in this system are the number of cars that each intersection lets through for each lane. The initial value for every lane is three cars.

There are 250 cars in the simulation. They pick their destination randomly and choose the shortest path to it. When they reach their destination, the process is repeated. In this scenario, the path between intersections A and B is chosen with a probability of $p = 0.9$. That means if a car starts in A (or B), it will choose the destination B (or A) with a probability of 90%, and the other five destinations are chosen with a probability of 2%. Similarly, the path between E_1 and E_2 is chosen with $p = .75$. Consequently, these two paths have the most traffic. The cars in the simulation are part of the environment.

Again, we introduce an artificial disturbance to analyse our metrics. For this purpose,

we block the central intersection C in the time frame from $t = 250$ to $t = 400$. During this time, intersection C is unavailable, and the cars have to use other paths. While during normal operation, the traffic between A and B flows through C, now the cars need to go through D_1 and D_2. Therefore, intersection D_2 will have to reconfigure its red-light times to handle both traffic streams.

Each intersection controller controls one signal light for every incoming lane. Since not all intersections have the same number of lanes, this system is heterogeneous, meaning the configuration vectors of the different intersection controllers differ. However, in this case, we can use the trick outlined in Chapter 4.5.3: since every signal light has its own configuration, we can consider the signal lights as the actual subsystems. This way, we get a collection of subsystems where every subsystem has the same dimension in its configuration vector.

Every connection consists of two lanes. Hence, we obtain a total of 14 subsystems in the simulation.

5.1.4 Artificial Road Network - Rush Hour

This scenario simulates rush hours in a city street network with two islands, each with a Manhattan-type network of sizes 3 by 5. The islands are connected by three bridges. The connections between two intersections provide one lane for each direction (see Figure 5.4).

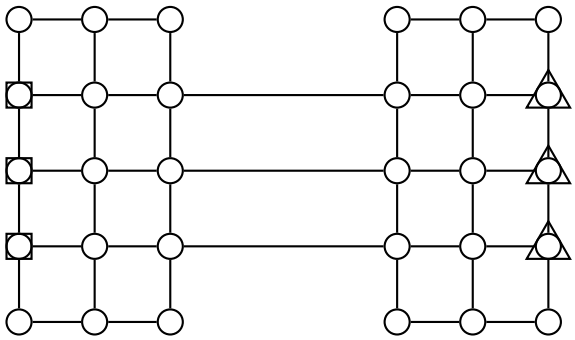


Figure 5.4: The street network for the rush hour scenario. The circles are intersections, and the lines represent the connecting streets. Every street has two lanes, one for each direction. The intersections with a square are homesteads, and the ones with triangles are workplaces.

During the whole simulation, 250 cars are driving around randomly. The behaviour of the cars (destination choosing) and the intersections (optimisation of waiting times) is the same as in the Road Block scenario described in Chapter 5.1.3.

Additionally, three intersections on one island are designated homesteads, and three intersections on the other are labelled as workplaces. Again, we introduce a disturbance

in the simulation. At $t = 250$, a total count of 500 new cars will appear randomly in the three homesteads and start driving towards one of the three workplaces. Therefore, the bridges have to handle an increasing flow from one side to the other in the following time steps. When the new cars arrive at their workplaces, they are removed from the simulation. At $t = 750$, this is repeated in the opposite direction.

To evaluate different sizes of the disturbance, we will analyse two more simulations with the same situation but with 300 extra cars during the rush hours in the first case and with only 100 additional cars in the second case.

Again, each signal light is considered an individual subsystem. Therefore, this scenario includes 94 subsystems.

5.1.5 Life-Like Road Network

While the previous two traffic scenarios were created to show the behaviour of the metrics under laboratory conditions, this scenario was created to bring the metrics closer to a real-world application.

In this scenario, we investigate simulated urban traffic based on real-world vehicle movements that were observed in the area of Lokstedt, an urban district of the city of Hamburg, Germany. Using the Aimsun Next traffic simulation software [11], a professional tool used by traffic experts and city planners, we model a part of Lokstedt with several main streets and 10 intersections (see Figure 5.5). For instance, the junction K1 corresponds to the real-world location $53.611563°N$, $9.955750°E$. Minor streets, as well as dead ends, are not part of the network model. The road sections are heterogeneous in length and width, with one, two or even more lanes per direction. So, vehicles can overtake or bypass smaller obstacles in parts of the network. The intersections K1, K2, K3, K5, and K6 are connected to *centroids*, which represent the surrounding network and let vehicles enter and leave the street sections.

The number of simulated vehicles is based on real-world observations collected on 03.07.2008. These traffic demands reflect the rush hour situation in the morning between 07:00 and 10:00 CET, as summarised in Table 5.1 for every hour.

Hour	Cars	Trucks
7:00	11905	479
8:00	12161	543
9:00	10755	570

Table 5.1: Hourly traffic demands across all sections of the Lokstedt simulation.

In contrast to the previous traffic scenarios, the traffic lights at the intersections are now controlled by the Organic Traffic Control system (OTC [214]). The OTC system adapts the durations of the traffic light phases in response to the current traffic demands

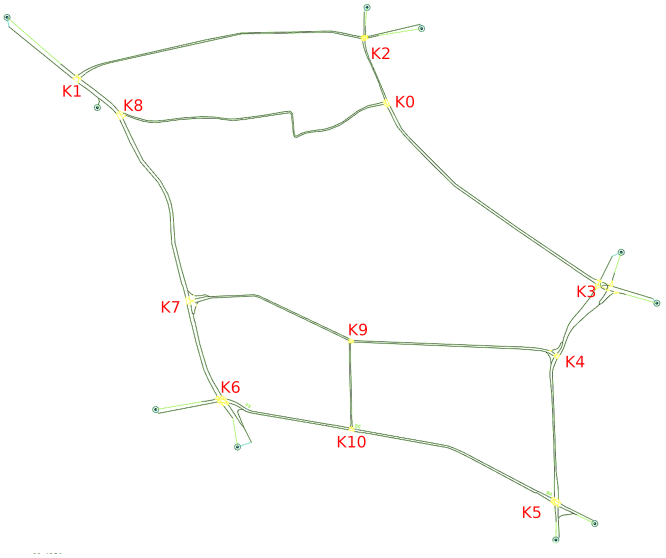


Figure 5.5: The simulated street network based on the street layout in Lokstedt/Hamburg, Germany.

with the aim of optimising the traffic flow and improving the vehicle throughput. For this, at each intersection, the CM of the OTC component analyses the current flow situation by creating statistics based on current sensor data and internal simulations. These statistics are updated every 5 minutes. Then, the CM selects suitable adaptations by applying the rules of a *Learning Classifier System*. If required, these rules are created using *Evolutionary Algorithms* together with an additional traffic simulator to evaluate the performance of these new rules. Since the performance evaluation and the other steps require an unpredictable amount of time, the subsystem update cycle time is not constant. Furthermore, the intersection can – if the feature is activated – communicate with each other using the *Decentralised Progressive Signal System* (DPSS, [230, 228]). In this case, the intersections exchange messages to optimise the traffic flow on a larger scale.

The scenario consists of three cases:

1. A simulation of the network with the originally recorded data without any further influences. Here, the DPSS is not active.
2. A simulation with the same conditions as before and additional disturbances. Here, a complete 15-minute blockage of the section from K5 to K10 is simulated at the beginning of each of the 3 hours. The opposite direction from K10 to K5 is not blocked, and this street allows vehicles to bypass the blockage (allowing for oncoming traffic).
3. A simulation with the same conditions and disturbances, and with activated DPSS.

The configuration we analyse in this case is the length of the green phase for every signal light. Counting each signal as an individual subsystem, we have a total of 54 subsystems.

5.1.6 Maritime Traffic

The Christian-Albrechts-Universität, where this thesis was supervised, is located in Kiel, Germany. Kiel is a seaport and end point of the Kiel Canal. Thus, the city sees much seaborne traffic. Therefore, we investigate a scenario which deals with traffic data from maritime vessels.

A wide range of maritime vessel types is required to regularly send out status and navigational information, including a unique identifier, the current location, speed, and course. This information is broadcasted with a standardised message format defined by the *Automatic Identification System* (AIS) [264].

The data-set [261] that we analyse contains AIS data recorded by AIS base stations in the area of responsibility of the *Danish Maritime Authority* (DMA)[260]. The data set includes data from ships in the area of the city of Kiel, Germany, and the Kiel Canal.

The Kiel Canal, one of the most travelled canals in the world, runs through northern Germany and connects the North Sea and the Baltic Sea [268]. On 30.11.2022 at 4:37 CET, a ship in the canal collided with a bridge, forcing the canal authority to stop all traffic through the canal.

The two main properties of a ship that conform to our notion of configuration are its course and its speed. The analysis of the speed data does not give any indicators for the canal accident. We assume that the reason for this is the general speed limit in the Kiel Canal (8.1 knots) and the Kiel seaport area (10 knots) and the fact that vessels equipped with AIS senders are usually larger vessels with slow changes in speed. Therefore, we will present the analysis for the *course over ground* data field from the AIS data provided by the DMA.

For comparison, we investigate two time frames. The first reaches from 28.11.2022 12:00 CET to 29.11.2022 12:00 CET and is used as the baseline. In this time frame, no remarkable events occurred. The second time frame contains the following 24 hours, from 29.11.2022 12:00 CET to 30.11.2022 12:00 CET. This includes the time of the accident. From the data set, those data points were selected that match the following criteria:

- The data point was recorded within the selected time frame.
- The location of the data point is in the *Kieler Förde* area (see Figure 5.6):
  - Latitude $\geq 54.309274°N$ and $\leq 54.432831°N$
  - Longitude $\geq 10.080272°E$ and $\leq 10.288947°E$
- The ship for the data point has at least one data point with a speed value > 0 within the time frame and the location boundaries.

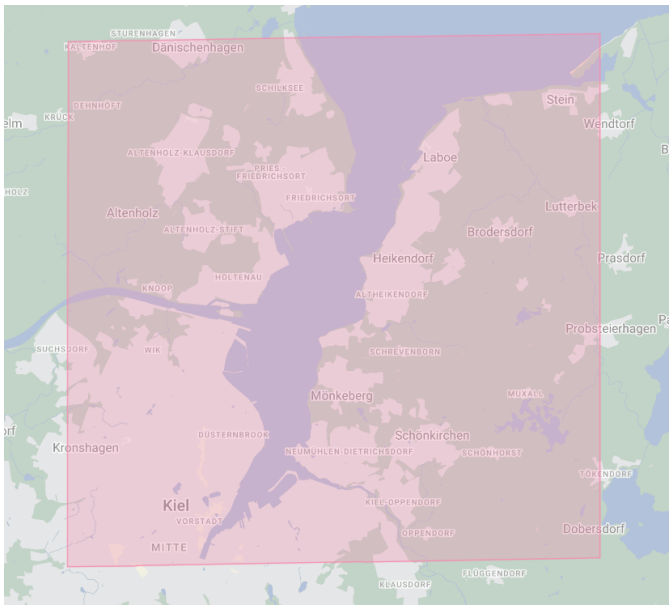


Figure 5.6: The Kieler Förde with the boundaries of the analysed AIS data points (red area). The Kiel Canal can be seen in the west, the Baltic Sea is in the northeast, and the city centre of Kiel is in the southwest. Map data © 2003 GeoBasis-de/BKG (©2009), Google [272].

In total, 14527 data points from 93 ships match the criteria for the first time frame, and 7864 data points from 74 ships fall in the second time frame.

In the data set, the *course over ground* is stored as an angle in the range $[0°, 360°)$, where 0° represents north. Our metrics require data for every time point. If no data is available in the data set for a given point in time, then the last seen value for the respective ship is used.

5.1.7 Game of Life

In this simulation, we have a toroidal cellular plane with 50x50 cells. Each cell has either the state of dead or alive, which is also the configuration we are observing. In every step, each cell computes the state it will have in the next step using the ”classical” rules depending on the number of living neighbours (cf. Chapter 2.2.3):

- It dies if fewer than two or more than three neighbours live.
- It stays alive if two or three neighbours live.
- It becomes alive if exactly three live neighbours are alive.

The simulation starts with a random initialisation and reaches, after 300 steps, a state with constant patterns and a few ”blinkers”. Figure 5.7 shows the simulation after 302 steps.

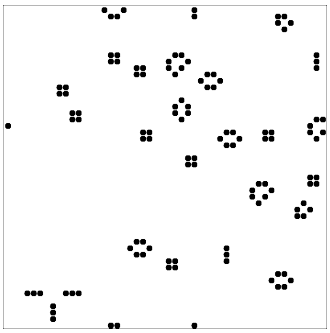


Figure 5.7: The Game of Life simulation at step $t = 302$.

For the configuration parameter, we will use the state of each cell. A living cell is represented by a 1, and a 0 denotes dead cells.

5.1.8 Taxonomy and Classification of the Scenarios

In this chapter, we will apply a taxonomy to the scenarios, show an additional classification, and explain why we chose them.

Taxonomical Dimensions of the Scenarios

In Chapter 2.2.2, we presented the self-adaptation taxonomies by *Rohr et al.* and *Krupitzer et al.*. We will now see, how can our scenarios be placed in these taxonomies.

Since the focus of this thesis is the analysis of configuration parameters, the taxonomical dimension of *Operation/Technique* is always the change of parameters. For the dimension of *Adaptation Control/Controller Distribution*, all scenarios use systems with a decentralised control structure. Furthermore, all systems apply the self-adaptation to the application layer. The dimensions *Activation/Time* and *Origin/Reason* are as follows:

- The rules of the Flocking scenario, especially the avoidance rule, lead to predictive self-adaptation. The birds change their direction if a collision is imminent. On the other hand, the introduced disturbance in this scenario leads to a reactive behaviour. Since this scenario has no interaction with any environment or user, the Origin of self-adaptation is the system itself.
- For the Smart Camera scenario, the source of self-adaptation is the environment because only if a camera tracks a person its angle changes. This change of angle propagates through the system via the overlap minimisation. This is a reactive behaviour.
- In the Roadblock scenario, the intersections perform their self-adaptation in reaction to the current volume of traffic. Although the source of the disturbance

lies within the system itself as one intersection becomes unavailable (i.e. internal system failure), the effective source of the self-adaptation remains the traffic. In this case, the taxonomical dimension of Origin/Reason is either the system or the environment, depending on the point of view.

- For the Rush Hour scenario, the dimensions are clear: the system reacts to a change in the environment.
- In the Life-Like Road Network scenario, the system is again primarily reacting to the environment. However, it contains an internal simulator to predict possibly better rule sets, which can lead to predictive self-adaptation.
- In the Maritime Traffic scenario, the system is comparable to the Flocking scenario. It contains predictive self-adaptation of internal origin (collision avoidance) and a reaction to the (environmental) canal closure.
- In the Game of Life scenario, the source for the adaptation is the system itself since it is self-contained and has no environment. The state changes are a reaction to the previous states.

Table 5.2 summarises the points.

Scenario	**Activation/Time**	**Origin/Reason**
Flocking	Predictive / Reactive	System
Smart cameras	Reactive	Environment
Roadblock	Reactive	Environment / System
Rush hour	Reactive	Environment
Life-Like Traffic	Predictive / Reactive	Environment
Maritime Traffic	Predictive / Reactive	System / Environment
Game of Life	Reactive	System

Table 5.2: Values of the taxonomical dimensions for the evaluation scenarios.

Further Classifications

The framework is intended to be applied to a wide range of SASO systems. Since the notion *wide range* is not very specific, we will investigate three more specific properties of SASO systems: *Openness*, *Size*, and *Closeness to Reality*:

- Openness applies to the environment and the system itself. A system operates in an open environment if new sources of influence can occur or existing can cease to exist. The system itself can be declared as open if the number of subsystems or other relevant components or structures (e.g. the control structure) change over time.
- Size means, in this case, simply the number of subsystems, which is a possible way to compare two SASO systems.

- Closeness to Reality is the distinction between highly artificial systems that were created for research purposes on one side and systems that are actually running in production environments on the other. Of course, this distinction is not binary. There are several stages between them.

Let us apply these three dimensions of comparison to our scenarios:

- **Flocking:** In this scenario, the environment and the system itself are closed. The number of birds does not change, and their environment is not altered over time as long as the shooter is considered present in the beginning. Compared to the rest of the scenarios, 50 birds is in the medium range of the scale. Although the scenario is a synthetic simulation on a torus, the flocking rules produce a behaviour close to natural phenomena. Therefore, we can place it halfway between an artificial and a real-world scenario.
- **Smart Camera Network:** With the same arguments as for the flocking simulation, this scenario has a closed system and a closed environment. With 10 subsystems, it is rather small. The abstract representation of the cameras and the setup of the museum and the events is highly artificial.
- **Artificial Road Network - Blocked Road:** This is, again, a closed scenario. Neither the number of cars nor the number of intersections change. With only 7 subsystems, it is the smallest scenario in our portfolio. As the name indicates, this is an artificial setup due to the network layout and the traffic model.
- **Artificial Road Network - Rush Hour:** In this case, the number of intersections still remains constant. But this time, the number of cars changes. This can be considered an open environment. With 30 subsystems, it is medium-sized. Like the previous traffic scenario, this is an artificial scenario.
- **Life-Like Road Network:** The system, with its constant number of intersections, is again closed. The environment, on the other hand, is very open. Vehicles enter and leave constantly. With 11 intersections, this system is a small one. Although the underlying real-world street network in this model does not contain any intelligent intersections, we can consider this scenario as very close to reality. The used traffic data and the network layout are based on real-world data, and the modelled intersection controllers have been implemented in real-world test beds.
- **Maritime Traffic:** The changing number of vessels in the observed area and the real-world environment make this system and the environment open. With the high number of 74 and 94 ships, this scenario is a large one. And, of course, this is a real-world scenario.
- **Game of Life:** In this closed scenario, we have 2500 subsystems. This makes it very large. As a cellular automaton, this is an artificial scenario.

Table 5.3 recaps the classification. The columns *System* and *Environment* refer to

the openness, and the column *Type* describes the closeness to reality. We need to note that the attributes in the column *Size* are only valid for comparing our scenarios with each other. A simulation with 2500 entities might be considered small in other contexts.

Scenario	**System**	**Environment**	**Size**	**Type**
Flockers	closed	closed	medium	artificial
Smart Camera Network	closed	closed	small	artificial
Road Network - Blocked Road	closed	closed	small	artificial
Road Network - Rush Hour	closed	open	large	artificial
Life-Like Road Network	closed	open	large	real-world
Maritime Traffic	open	open	large	real-world
Game of Life	closed	closed	very large	artificial

Table 5.3: Classification of the scenarios with respect to system openness & environment openness, size and type of closeness to reality.

The Reasons for Selection

The scenarios are chosen such that their entirety covers most of the aforementioned taxonomical characteristics and dimensions of quality. Additionally, some profane reasons apply:

- Flocking is the prime example of systems that are inspired by natural phenomena and, at the same time, very simple. We included it in our evaluation because it is totally decentralised and fully defined by very few rules. Furthermore, it can be found in most lectures and books about multi-agent systems and emergence.
- The Game of Life scenario is included in the evaluation because it plays a prototypical role in cellular automata and emergence throughout literature. Cellular automata can be a starting point for building SASO systems [109].
- The smart camera scenario was explicitly designed to show that each of our metrics alone has limitations.
- We chose the traffic scenarios for evaluation because we believe that intelligent traffic management is an essential technology for the future of transportation, especially in light of the climate crisis.

In the Flocking scenario and in the Rush Hour scenario, we have included simulations with different strengths of the disturbances to analyse how the metrics react to such deviations in strength. These are the only two scenarios where we do this for the following reasons:

- The strength of disturbance in the Smart Camera scenario is explicitly tailored to this scenario.
- In the Road Block scenarios (artificial and Life-Like), there is a structural disturbance. Moving the road block to another position or adding additional blockades

would lead to a different scenario.
- In the Maritime Traffic Scenario, the core of the analysis is the fact that we analyse real-world data.
- The Game of Life scenario has no disturbances.

5.2 Results

In this chapter, we will show how to apply the metrics to the scenarios above in the order of *Entropy, Configuration Stability, Configuration Variability, Coherence, Parameter Usage* and *Configuration Divergence*. Due to the system design, only the Life-Like Road Network scenario provides communication data. In the other two traffic simulations, all intersections work independently and do not incorporate information from their neighbours. Therefore, there is no communication present. The communication between ships is not available for the Maritime Traffic scenario. In general, ship radio messages are not stored. Only the documentation of radio communication in a log book is required [257]. For the two remaining scenarios, Game of Life and Flocking, we will create an artificial communication model to evaluate the *SO Divergence* metric.

Furthermore, in each scenario, we will show the influence of the tuning parameters for the according metrics. After the presentation of the results, we will close each scenario with a short discussion of the outcomes.

5.2.1 Results for the Flocking Scenario

Here, we provide the analysis for the metrics in the Flocking scenario with the bigger disturbance. The radius of the disturbance is 50 units, and its length is 3 time steps. This section aims to show the influence of the parameters on the metrics. The results in the cases with the less significant disturbance are presented in Chapter 5.2.2. For the analysis in the current case, our attention is mainly turned to the steps with $t \geq 500$ where the disturbance occurs.

Entropy

For the application of the *Entropy*, we need to fix a number M of bins for the estimation. Since we deal with angles in this scenario, a fraction of 360 seems appropriate. Figure 5.8 shows the *Entropy* for several values of M. We see that the higher values isolate details more clearly (e.g. the small plateau from $t = 267$ to $t = 290$). Yet, all three graphs deliver the same notable spikes. Especially the disturbance at $t = 500$ is clearly visible due to the extreme jump in the time series, which does not occur at other steps with such an amplitude.

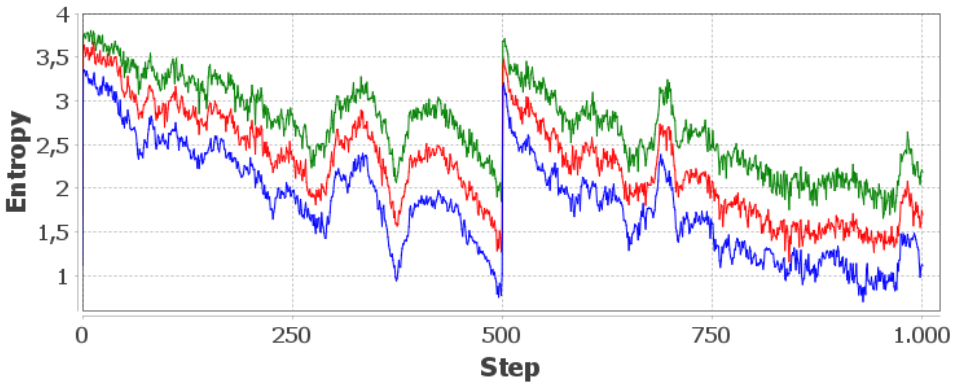


Figure 5.8: The time series of the Entropy for the Flockingscenario with bin counts $M = 45$ (blue), $M = 90$ (red), and $M = 180$ (green). The disturbance happens at $t = 500$.

Configuration Stability

Since the calculation of the *Configuration Stability* requires the distributions of the configuration parameter (i.e. the angle of orientation) of the bird during different time frames, we will look at an example. Figure 5.9 shows the directions of an affected bird. The sudden change of orientation at $t = 500$ is clearly visible. Shortly before the disturbance, the bird was flying with an orientation angle close to 3. With the disturbance, the orientation changed to a value near 0.5. The gaps between $t = 240$ and $t = 450$ are the result of a wrap-around at the $\pm\pi$ border. The graph is continuous at these points.

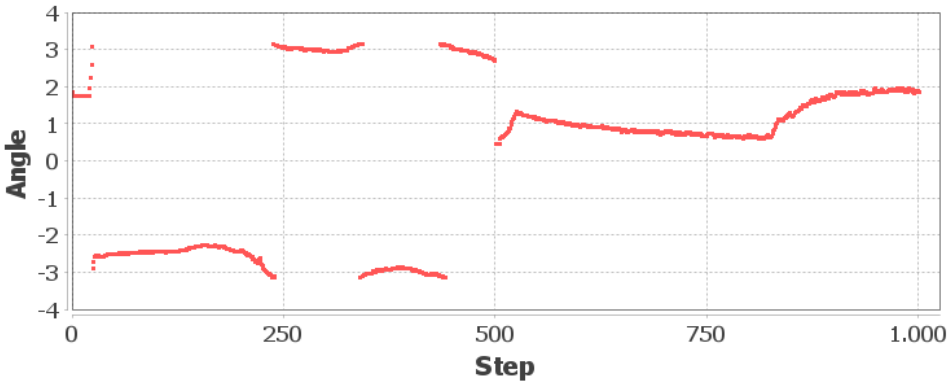


Figure 5.9: The angles of directions of one bird that is affected by the disturbance at $t = 500$.

Using two time windows of size $M = 10$ steps and a lag of $L = 10$ steps, we can apply the density estimation and receive a result that is shown in Figure 5.10. The shift

of angles from 3 to 0.5 is reflected by the according maxima in the densities.

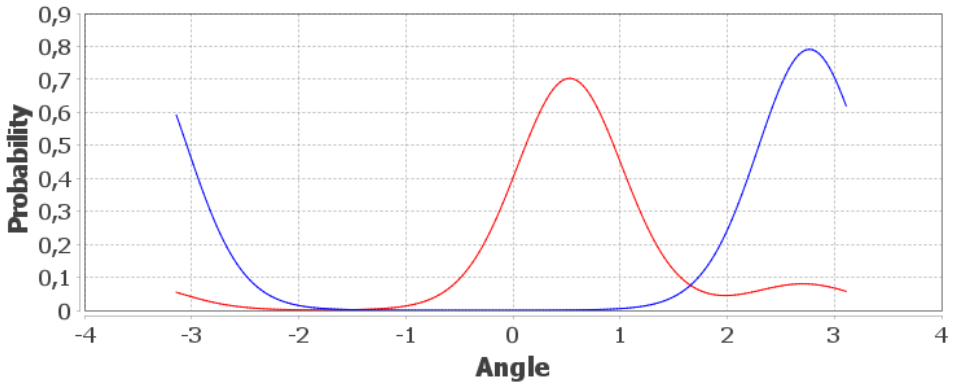


Figure 5.10: The densities of the angles from Figure 5.9 in the time frames $t = 490$ to $t = 500$ (blue) and from $t = 500$ to $t = 510$ (red).

For the application of the *Configuration Stability*, we need to find suitable parameters for the window sizes M, L and the threshold ε. Figures 5.11, 5.12 and 5.13 show the influence of the parameters on the result.

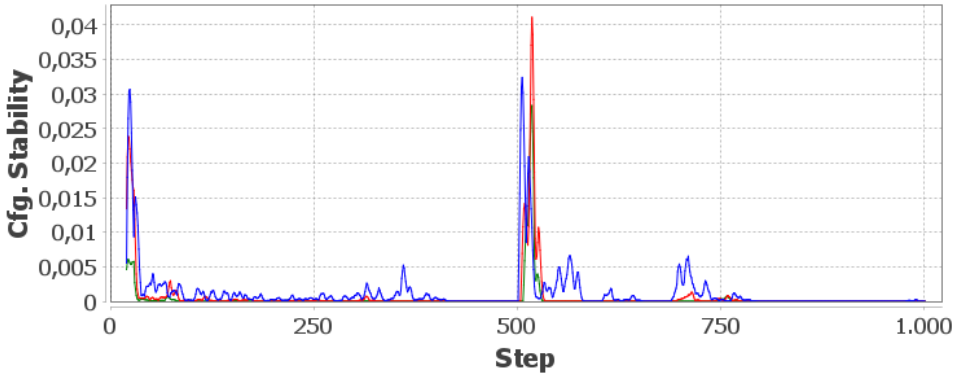


Figure 5.11: The influence of the tuning parameters in the Flocking scenario for the *Configuration Stability* for $M = L = 10$. The threshold ε is 0.1 (blue), 1.0 (red), and 2.0 (green). The disturbance happens at $t = 500$.

We see that in this scenario, smaller thresholds ε create noisier graphs with several peaks that might even be interpreted as false positives. For $M = L = 30$ and $\varepsilon = 0.1$, the impact of the disturbance even gets lost in the general noise of the graph. In this case, the peaks aside the disturbance (e.g. at $t = 750$) are notably higher than the maximum value in the time after the disturbance. Bigger thresholds, on the other hand, can suppress actual interesting events. For example, the parameters $M = L = 20$ and $\varepsilon = 0.1$ show a strong reaction to the simulation's dynamics near $t = 730$, while for $\varepsilon = 2$,

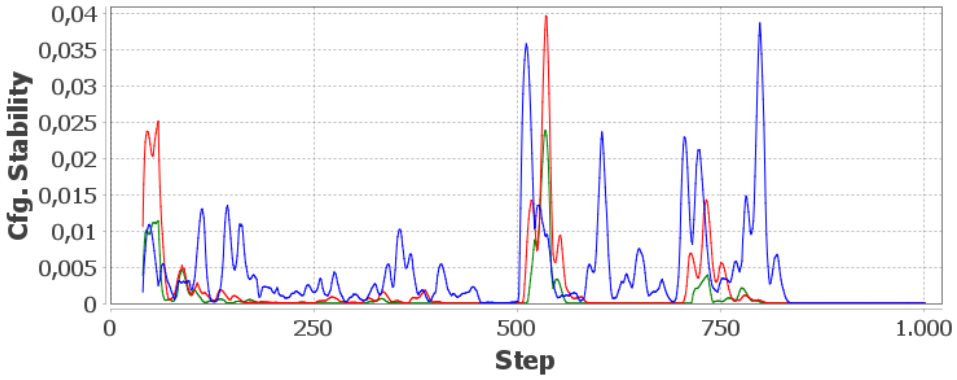


Figure 5.12: The influence of the tuning parameters in the Flocking scenario for the *Configuration Stability* for $M = L = 20$. The threshold ε is 0.1 (blue), 1.0 (red), and 2.0 (green). The disturbance happens at $t = 500$.

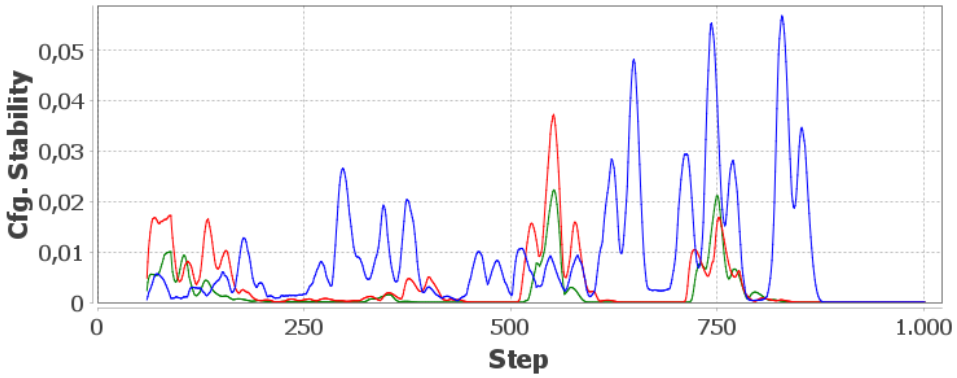


Figure 5.13: The influence of the tuning parameters in the Flocking scenario for the *Configuration Stability* for $M = L = 30$. The threshold ε is 0.1 (blue), 1.0 (red), and 2.0 (green). The disturbance happens at $t = 500$.

this peak starts to vanish. Bigger window sizes lead to smoother graphs but also delay the detection of peaks.

Table 5.4 summarises how the different parameters influence the detection of the disturbance after $t = 500$. It lists the parameters M, L and ε, the maximum value and the time point/location of the peak corresponding to the disturbance, and where in Figures 5.11, 5.12 and 5.13 it is shown.

The heuristic from Chapter 4.3.9 gives us $M = L = 10$ and $\varepsilon = 1$ as suitable parameters. This combination gives us the times series for the *Configuration Stability*, as shown in Figure 5.14. The impact of the disturbance is obvious, but it only becomes visible with a delay. The new orientations need some time steps to sufficiently affect the distribution of angles within the time frame such that the divergence with the previous

M	L	ε	**Value**	**Location**	**Figure**
10	10	0.1	0.032	$t = 506$	5.11 blue
10	10	1.0	0.041	$t = 518$	5.11 red
10	10	2.0	0.028	$t = 518$	5.11 green
20	20	0.1	0.036	$t = 511$	5.12 blue
20	20	1.0	0.040	$t = 535$	5.12 red
20	20	2.0	0.024	$t = 534$	5.12 green
30	30	0.1	no peak	no peak	5.13 blue
30	30	1.0	0.037	$t = 552$	5.13 red
30	30	2.0	0.022	$t = 552$	5.13 green

Table 5.4: The influence of the tuning parameters for the *Configuration Stability* on the location and height of the peaks after the disturbance in the Flocking simulation.

time frame exceeds the threshold.

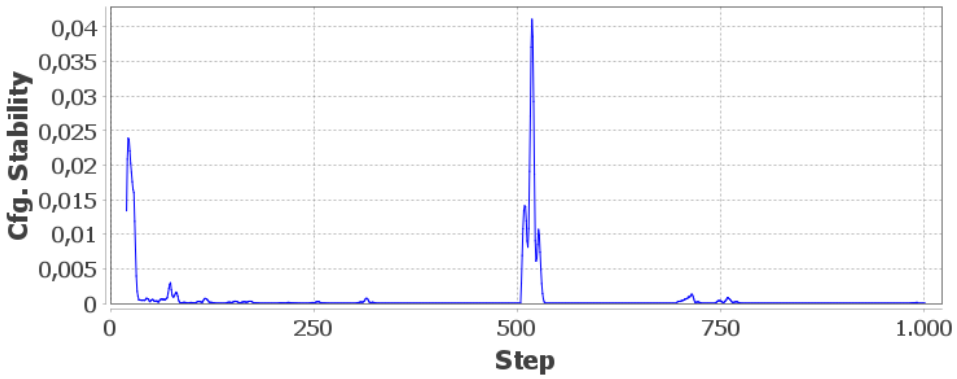


Figure 5.14: The time series of the *Configuration Stability* for the Flocking scenario with $M = L = 10$ and $\varepsilon = 1$. The disturbance happens at $t = 500$.

Configuration Variability

In Figure 5.15, we see the time series of the *Configuration Variability*. Again, the disturbance creates a strong change in the time series and a peak of $c_v \approx 0.47$ at $t = 502$. However, other time steps also show notable changes, such as a steep rise at $t = 380$ or the drop between $t = 566$ and $t = 582$. Furthermore, we see relatively low values of $c_v \approx 0.02$ shortly before the disturbance. This, again, is the result of the increasing alignment of the birds, which leads to decreasing distances for the data points from their respective cluster centroids during the clustering.

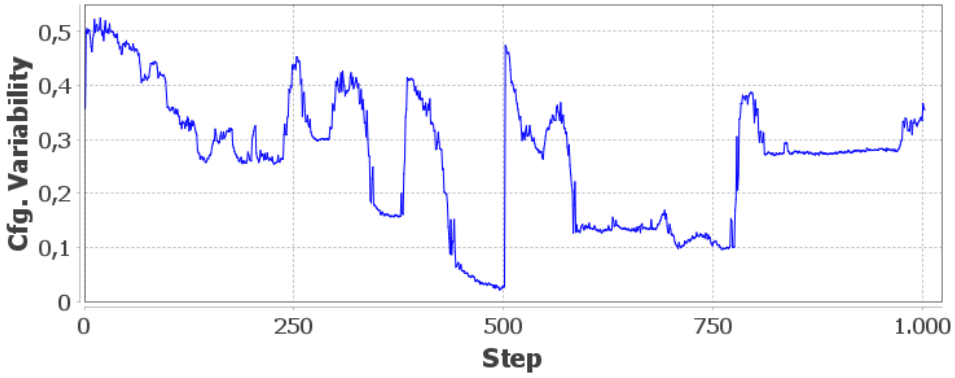


Figure 5.15: The time series of the *Configuration Variability* for the Flocking scenario. The disturbance happens at $t = 500$.

As mentioned at the beginning of this chapter, we will take a look at the differences between several clustering algorithms that can be used in the *Configuration Stability*. Figure 5.16 shows a detailed view of the time series for the G-Means [86], K-MeansPDN [171], and the X-Means algorithms. As we can see, all three algorithms lead to nearly identical results in the metric.

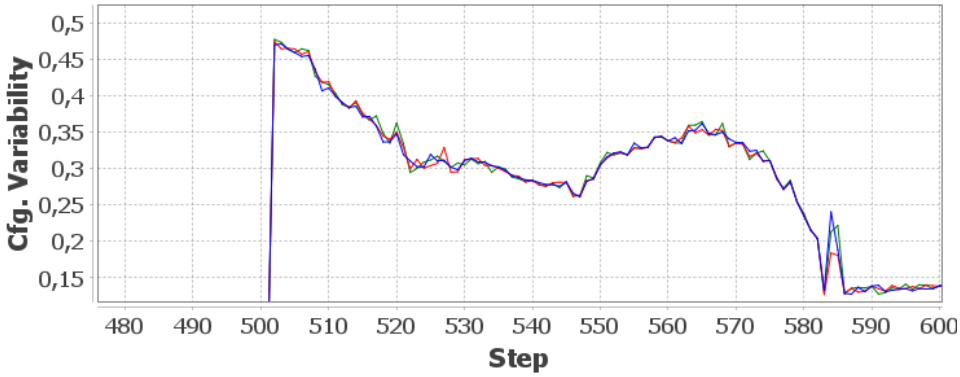


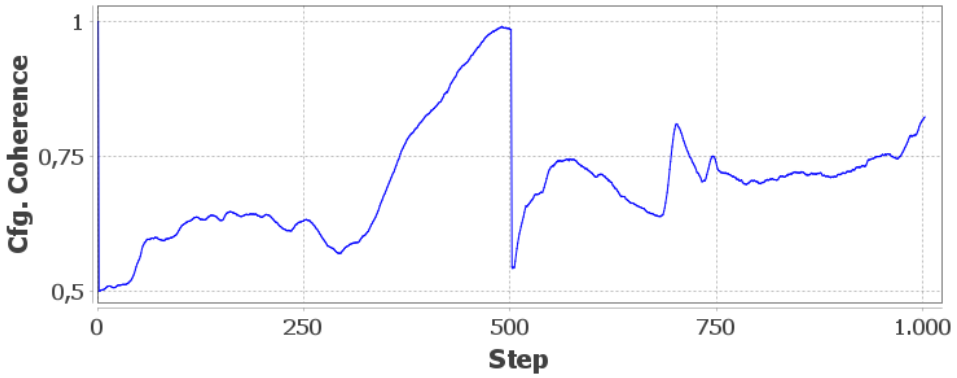
Figure 5.16: Details of the time series of the *Configuration Variability* for different cluster algorithms. Blue: X-Means, red: G-Means, green: K-MeansPDN.

Coherence Metrics

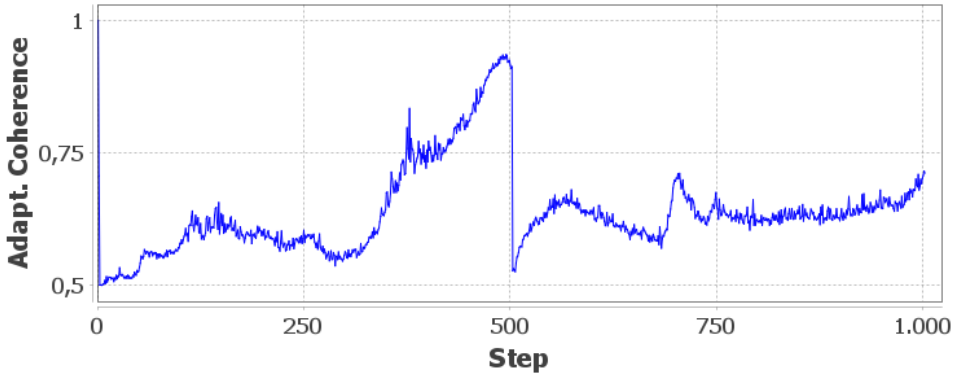
Figure 5.17 shows the result for the *Coherence* metrics. As a result of the alignment rule, the birds tend to fly more and more in the same direction. So, the angles of orientation decrease in difference. That leads to an increasing coherence with a maximum *Configuration Coherence* value of $coh_c \approx 0.93$ at $t = 490$. Then, the disturbance at $t = 500$ brings many new orientations into the flock and, therefore, induces the sharp

drop in the coherence values. Since this kind of drop does not occur at any other step, both Coherence metrics give a clear indication of an unusual event.

The *Adaptation Coherence* gives a noisier graph because fewer birds are considered in the calculation.



(a) Configuration Coherence



(b) Adaptation Coherence

Figure 5.17: The time series of the Coherence metrics for the Flocking scenario. The disturbance happens at $t = 500$.

Parameter Usage Metrics

Figure 5.18 depicts the *Global Parameter Usage* with different window sizes. In this case, the different window sizes only lead to longer times where the respective maximum interval length of parameter values is persistent. Again, the disturbance is clearly visible in the graph with the jump from $U_g \approx 0.1$ to $U_g \approx 0.75$. After $t = 500$, a wider range of values is used, but this range does not increase after that. So, the graph for the time window $M = 20$ still includes effects at $t = 520$ that happened at $t = 501$.

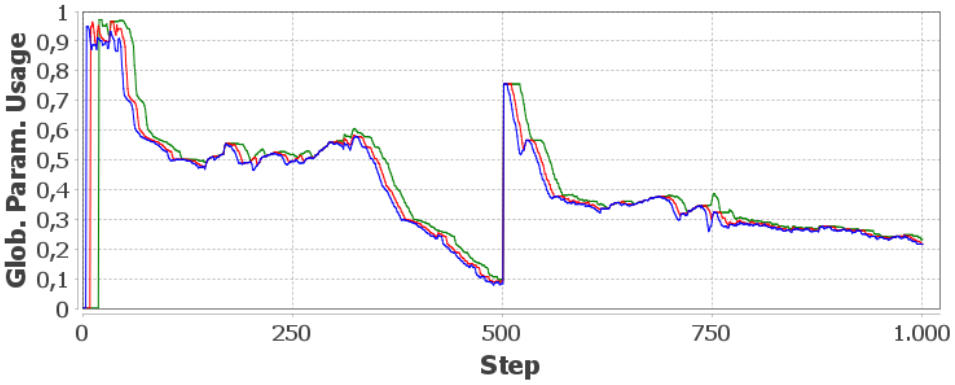


Figure 5.18: The time series of the Global Parameter Usage for the Flocking scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The disturbance happens at $t = 500$.

Figure 5.19 shows the *Average Parameter Usage* with different window sizes. Again, longer time windows lead to longer influences of events on the graph. The plateau beginning at $t = 500$ has a width of M steps and a height of $U_a \approx 0.175$ for $M = 20$ and $U_a \approx 0.164$ for $M = 5$.

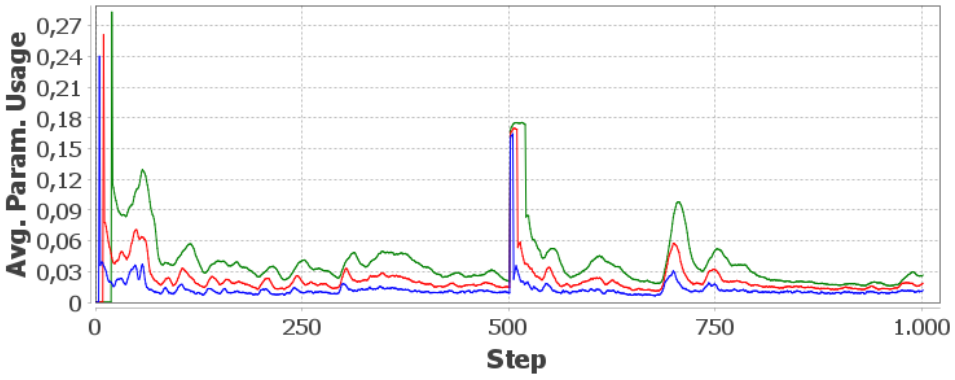


Figure 5.19: The time series of the *Average Parameter Usage* for the Flocking scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The disturbance happens at $t = 500$.

Another effect of longer time windows are higher values at certain time steps that may lead to false positive interpretations of unusual adaptation events. Here, an example is $t = 705$ for $M = 20$. Also, the disturbance gives a more prominent spike in the graph than in the graphs for the *Global Parameter Usage*.

The actual implementation of the metric assigns a 0 to the metric until the first time frame has reached the desired length. Therefore, we have the vertical lines at $t = M$.

Configuration Divergence

In Figure 5.20, we see the *Configuration Divergence* with different values for the lag parameter L. Higher values lead, in this case, to wider peaks and higher values for secondary ones. The value for the main peak at $t = 505$ stays at $c_d \approx 1.86$ for $M = 5$ and changing L. The baseline is reached after L steps in this case.

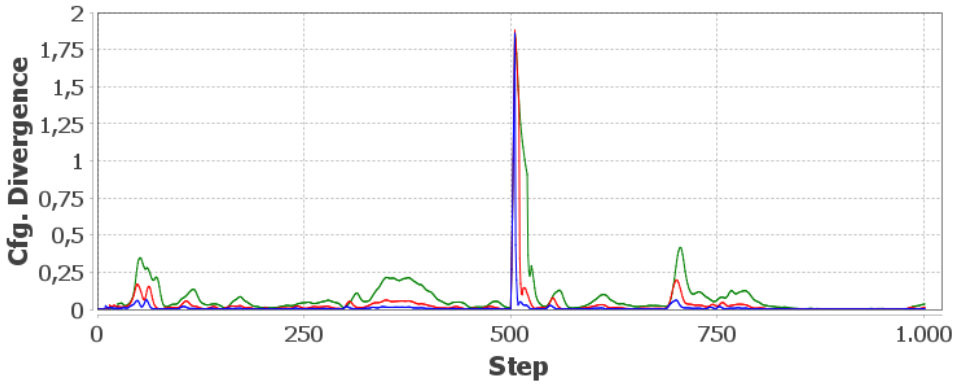


Figure 5.20: The time series of the *Configuration Divergence* for the Flocking scenario with parameters $M = 5$ and $L = 5$ (blue), $L = 10$ (red), and $L = 20$ (green). The disturbance happens at $t = 500$.

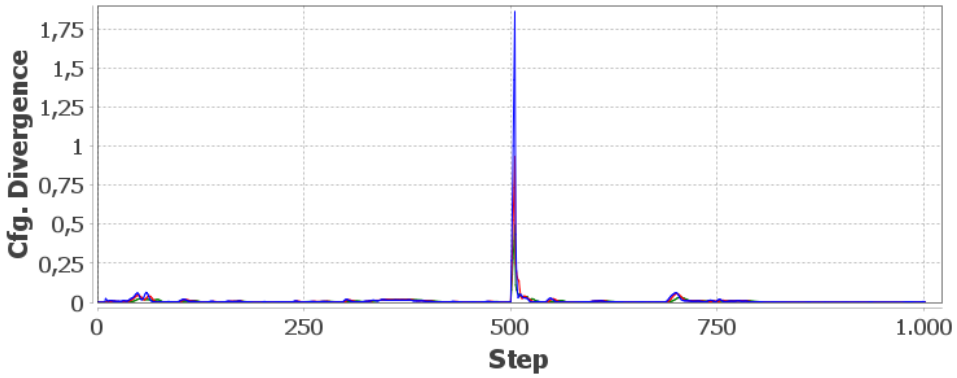


Figure 5.21: The time series of the *Configuration Divergence* for the Flocking scenario with parameters $L = 5$ and $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The disturbance happens at $t = 500$.

Figure 5.21 shows the influence of the window size M on the *Configuration Divergence*. Here, higher values lead to smaller values for the peaks. For $M = 20$, the peak reaches $c_d \approx 0.495$, while $M = 10$ gives $c_d \approx 0.930$. With $M = 5$, we see the peak value of $c_d \approx 1.86$. All peaks are at $t = 505$.

Self-Organisation Divergence

Although this simulation does not implement direct communication between the birds, we can state that a kind of message exchange is present: the birds broadcast their current course. This means that at every time step, every bird sends out a message containing its configuration. Thus, at any given time, the totality of messages is equal to the joint configuration of the system. Therefore, the *Self-Organisation Divergence* is equal to the *Configuration Divergence*.

Summary

The results show that in this scenario, the disturbance has a substantial impact on the time series of all applied metrics. The affected time series react with an extremely steep change. For the *Configuration Stability* (with suitable parameters), *Configuration Divergence* (with all of the shown parameters), and the *Average Parameter Usage*, the disturbance even creates an isolated, prominent peak in the graph. On the other hand, the *Configuration Variability* shows additional time points with strong changes. In the *Entropy*, the disturbance is the only point where a steep rise occurs, but the graph is noisy and shows several other peaks.

5.2.2 Results for the Flocking Scenario – Small Disturbances

We will now investigate how the metrics behave with a less significant disturbance. Using the parameter values that turned out suitable, we will analyse the Flocking scenario with a disturbance of radius 25 units and a length of one time step. For short, we will call this disturbance $D_{25,1}$. The second analysed disturbance $D_{10,1}$ has radius 10 units and lasts, again, one time step. We will call the third, most significant disturbance $D_{50,3}$, which we analysed above, and use it as a reference.

As expected, all metrics show the same behaviour in all three disturbance cases in the first 499 steps. Due to the different number of affected birds, the second half of the simulations each show a different behaviour of the flock. Thus, we will now focus on the few time steps after the disturbance.

Entropy

The bin count M is set to $M = 45$ for this analysis. The results are shown in Figure 5.22. As in $D_{50,3}$, the *Entropy* rises immediately in the other two simulations. In $D_{25,1}$, it reaches the same value of $H \approx 3.25$ at $t = 502$. For $D_{10,1}$, this peak reaches its maximum of $H \approx 2.49$ at $t = 503$. Although the peak in $D_{10,1}$ is not as high, it is still extreme with its jump from $H < 1.0$ within three steps.

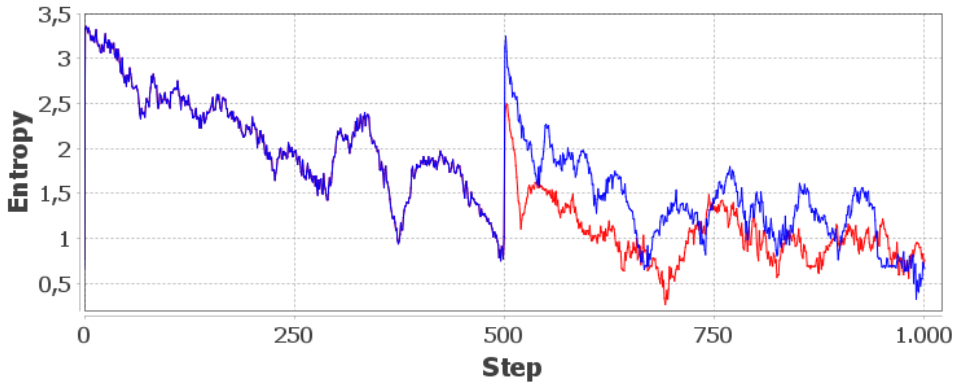


Figure 5.22: The time series of the *Entropy* for the Flocking scenario with bin count $M = 45$. Blue shows $D_{25,1}$, and red is $D_{10,1}$. The disturbance happens at $t = 500$.

We see that the significance of the disturbance has an influence on the metric. Nevertheless, the Entropy can detect the small impact of $D_{10,1}$.

Configuration Stability

For the *Configuration Stability*, we use the window parameters $M = L = 10$. Using $\varepsilon = 1$, as we did above for $D_{50,3}$, the metric reacts to the disturbance in $D_{10,1}$ with an extremely small peak of $c_s \approx 0.0001$. With that configuration, the *Configuration Stability* creates much higher peaks for other events in the first 300 steps, as we can see in Figure 5.23.

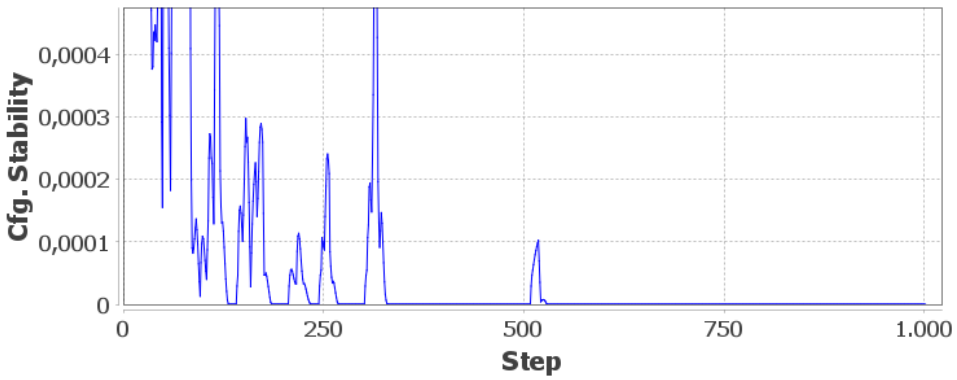


Figure 5.23: The time series of the *Configuration Stability* for the Flocking scenario with $M = L = 10$ and $\varepsilon = 1$ for the case $D_{10,1}$. The disturbance happens at $t = 500$.

However, changing the threshold parameter to $\varepsilon = 0.1$ allows the metric to identify the event. In Figure 5.24, we see the comparison of $D_{25,1}$ and $D_{10,1}$, where $D_{10,1}$ is configured with $\varepsilon = 0.1$. The peak in the metric for $D_{25,1}$ reaches $c_s \approx 0.048$ at $t = 518$,

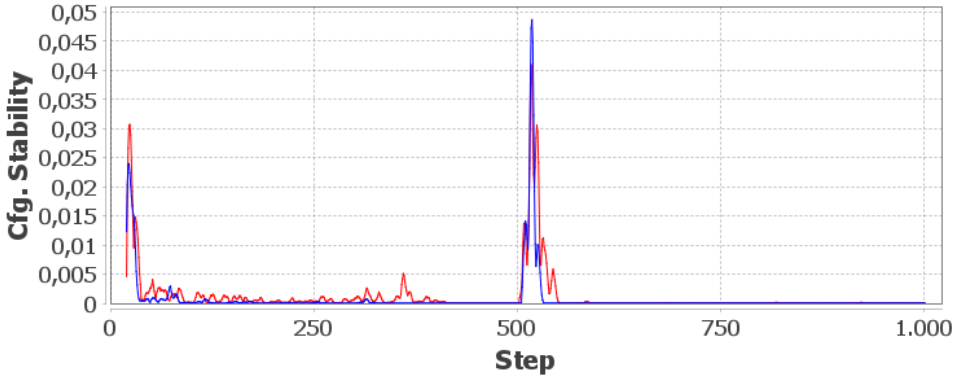


Figure 5.24: The time series of the *Configuration Stability* for the Flocking scenario with $M = L = 10$ and $\varepsilon = 1$ for the case $D_{25,1}$ (blue) and $M = L = 10$ and $\varepsilon = 0.1$ for the case $D_{10,1}$ (red). The disturbance happens at $t = 500$.

while $D_{10,1}$ gives $c_s \approx 0.041$ at $t = 517$. Compared to the previous simulation with $D_{50,3}$, where the metric resulted in a peak with $c_s \approx 0.041$ at $t = 517$, we obtain the same level of reaction at nearly the same points in time.

Configuration Variability

In Figure 5.25, we see the time series of the *Configuration Variability*. The simulation with the disturbance $D_{25,1}$ creates a peak at $t = 503$ with $c_v \approx 0.47$. This is the same value as for $D_{50,3}$. On the other hand, $D_{10,1}$ leads to a smaller peak with $c_v \approx 0.34$, again at $t = 503$. Since the value at $t = 500$ is $c_v \approx 0.026$, the reaction is still immediate and strong in both cases.

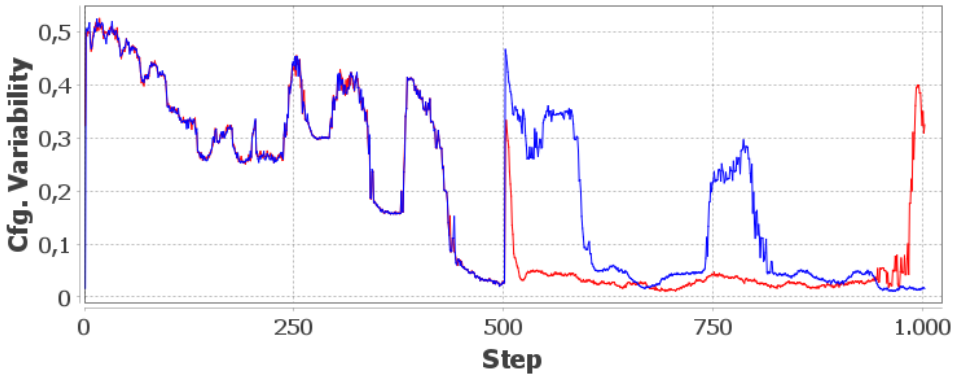


Figure 5.25: The time series of the *Configuration Variability* for the Flocking scenario in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

Coherence Metrics

Figures 5.26 and 5.27 show the results for the two *Coherence* metrics for the two smaller disturbances. Both disturbances lead to the same reaction pattern as for $D_{50,3}$, only with a different strength.

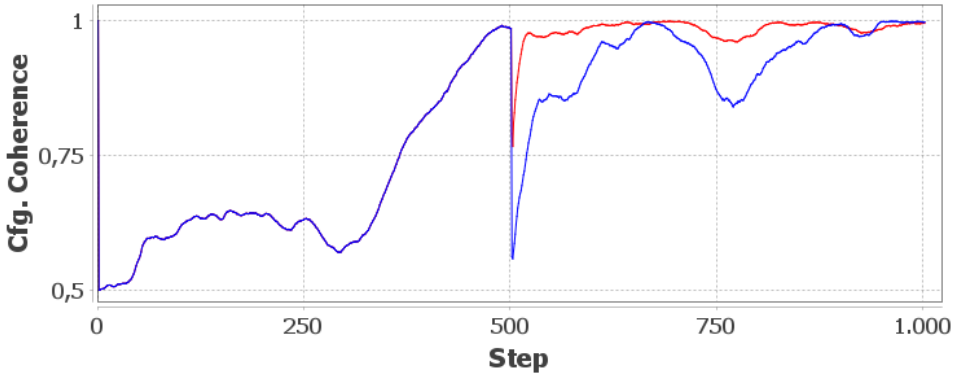


Figure 5.26: The time series of the *Configuration Coherence* for the Flocking scenario in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

The *Configuration Coherence* drops from $coh_c \approx 0.99$ to $coh_c \approx 0.56$ at $t = 502$ in the $D_{25,1}$. This is nearly the same level as in $D_{50,3}$, where the metric fell to $coh_c \approx 0.54$. For $D_{10,1}$, the decline is smaller and only reaches $coh_c \approx 0.77$ at $t = 502$.

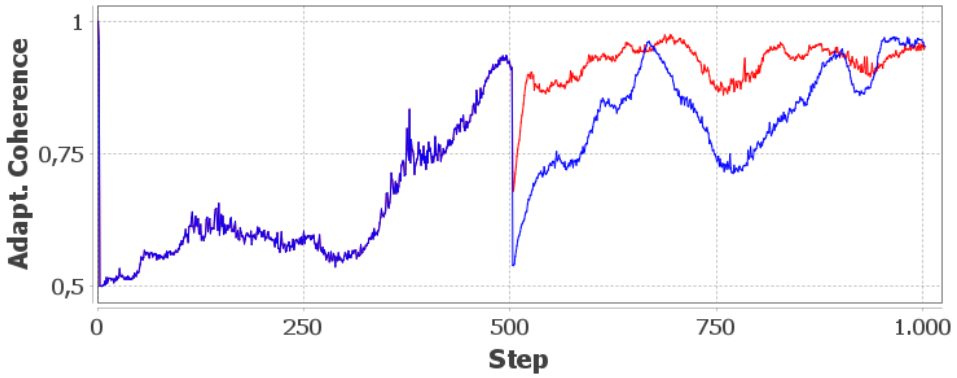


Figure 5.27: The time series of the *Adaptation Coherence* for the Flocking scenario in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

The *Adaptation Coherence* shows a comparable reaction. The smallest disturbance $D_{10,1}$ shows the smallest drop, with $coh_a \approx 0.68$ at $t = 504$. The next bigger disturbance $D_{25,1}$ results in $coh_a \approx 0.54$ at $t = 504$, which is, again, nearly the same level as in $D_{50,3}$, where the drop reached $coh_a \approx 0.53$ at $t = 504$.

Parameter Usage Metrics

Figure 5.28 shows the *Global Parameter Usage* for the two minor disturbances. Again, the different strengths of the disturbances lead to different heights of the peaks. Both $D_{50,3}$ and $D_{25,1}$ give a value of $U_g \approx 0.75$ at $t = 502$. The smaller disturbance leads to a peak with $U_g \approx 0.64$ at $t = 502$.

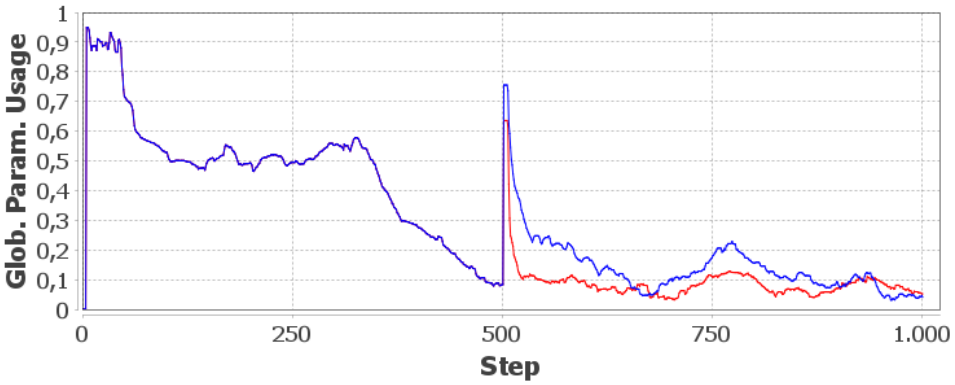


Figure 5.28: The time series of the *Global Parameter Usage* for the Flocking scenario with window size $M = 5$ in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

Figure 5.29 presents the *Average Parameter Usage*. Here, the influence of $D_{10,1}$ with $U_a \approx 0.078$ at $t = 505$ is only half as strong as the one of $D_{25,1}$, where we get a value of $U_a \approx 0.154$ at the same time step. Again, this level is comparable to $D_{50,3}$, where the spike at $t = 505$ reaches $U_a \approx 0.164$.

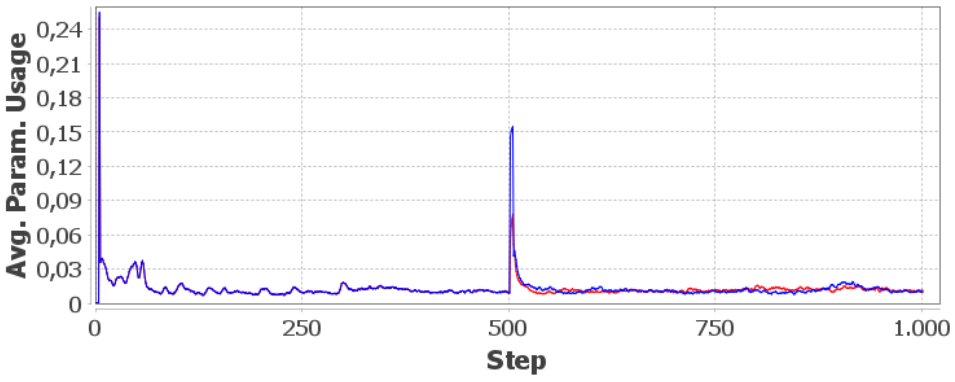


Figure 5.29: The time series of the *Average Parameter Usage* for the Flocking scenario with window size $M = 5$ in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

Configuration Divergence

In Figure 5.30, we see the *Configuration Divergence*. The peak in this metric is at $t = 505$ for all three disturbances. While the two bigger disturbances $D_{50,3}$ and $D_{25,1}$ lead to a comparable impact in the other metrics, the *Configuration Divergence* is more sensitive. In $D_{50,3}$, the peak value is $c_d \approx 1.86$. In $D_{25,1}$, it is only $c_d \approx 1.50$ and in $D_{25,1}$ even only $c_d \approx 0.25$. Since the value for the metric never exceeds $c_d = 0.1$ outside the disturbances, the reaction is still very clear in the smaller cases.

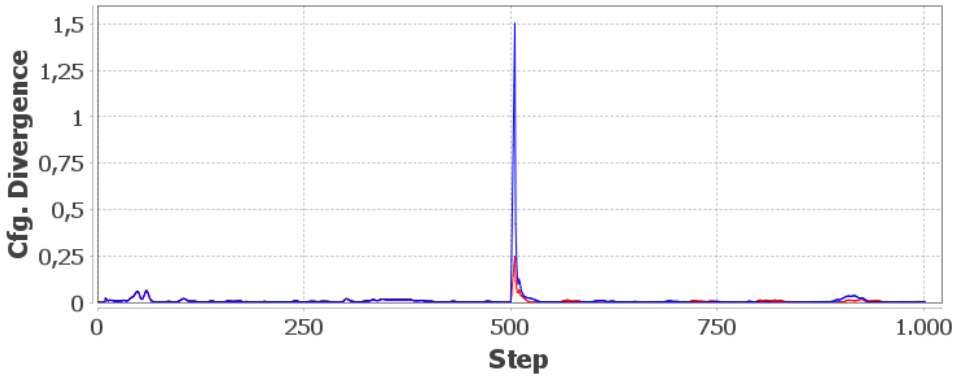


Figure 5.30: The time series of the *Configuration Divergence* for the Flocking scenario with parameters $M = L = 5$ in the cases $D_{25,1}$ (blue) and $D_{10,1}$ (red). The disturbance happens at $t = 500$.

Summary

In this scenario, the application of the metrics shows that the strength of the disturbance has a notable impact on the resulting spikes in the graphs. Only the *Configuration Stability* requires a reconfiguration of its parameters. However, the new parameter set is usable in all three disturbance setups, as Figures 5.11, 5.12, 5.13 and 5.24 show. Except for the *Configuration Divergence*, all metrics show a reaction on the same level to the two bigger disturbances $D_{50,3}$ and $D_{25,1}$, although $D_{50,3}$ influences more birds and lasts three times longer. The *Configuration Divergence*, on the other hand, shows different value levels for all three disturbances. The difference between $D_{10,1}$ and $D_{25,1}$ is individual to each metric. While *Configuration Divergence* shows a difference of factor 6 in these cases, the Global Parameter Usage differs only by 17%. In summary, all metrics can identify all three disturbances but react with different value levels.

5.2.3 Results for the Smart Camera Scenario

This section presents the results of the Smart Camera scenario. We compare the disturbed and the undisturbed scenario with a particular focus on the disturbance at $t \geq 110$.

Entropy

We start with a comparison of different values for the bin count M in the *Entropy* metric. Again, the configuration is a value for angles. Thus, a divisor of 360 appears natural. First, Figure 5.31 shows the difference between $M = 45$ and $M = 180$ for the undisturbed simulation. Figure 5.32 gives the results for $M = 90$. We see that with $M = 180$, several spikes are filtered out, e.g. the drops at $t = 100$ and $t = 150$. The difference between $M = 45$ and $M = 90$ shows only in the width of the spikes. For $M = 45$ the spike at $t = 100$ has a width of five steps, while its size shrinks to three steps for $M = 90$.

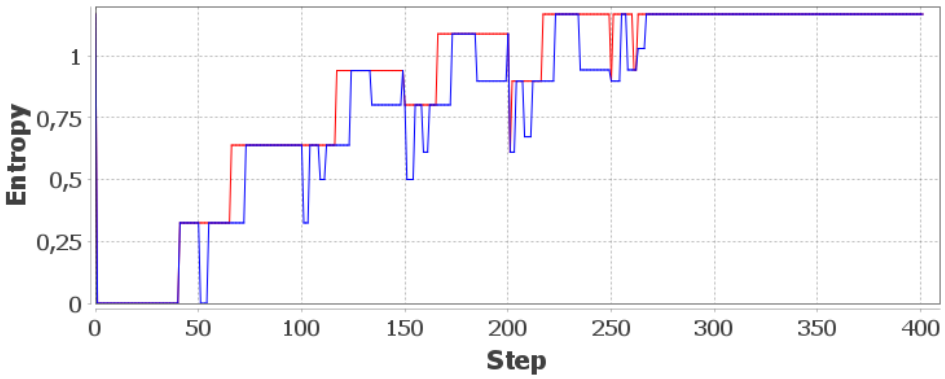


Figure 5.31: The time series of the *Entropy* for the Smart Camera scenario with bin counts $M = 45$ (blue) and $M = 180$ (red) in the undisturbed case.

Furthermore, Figure 5.32 depicts the comparison between the disturbed and the undisturbed simulation. We see that the disturbance leads to a sudden rise in the entropy at $t = 111$ and a higher base level value around $H \approx 1.5$ in the following steps. The undisturbed scenario does not exceed a value of $H \approx 1.2$.

Due to the strong change in the time series, the *Entropy* metric can detect this event. In direct comparison with the undisturbed case, which acts as an expectation for the time series, the influence of the attack becomes obvious.

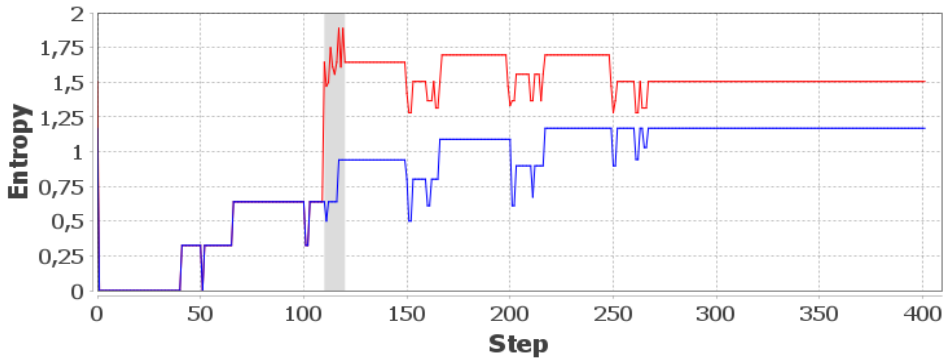


Figure 5.32: The time series of the *Entropy* for the Smart Camera scenario with bin count $M = 90$. The blue graph shows the undisturbed simulation, and the red graph the hacked scenario. The duration of the attack is marked grey.

Configuration Stability

In this scenario, the threshold parameter ε that distinguishes active and inactive subsystems in the *Configuration Stability* metric is notably different from the value for the Flocking scenario. While in the latter, we used $\varepsilon = 1$, we will need one magnitude smaller in this scenario. Here, the metric gives a constant zero for $\varepsilon = 1$.

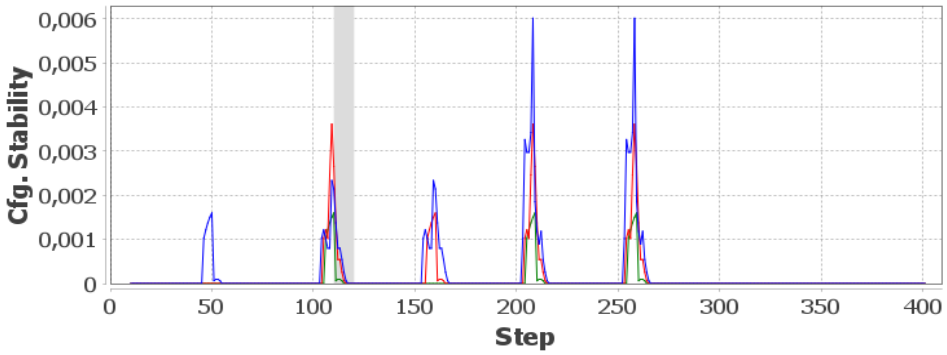


Figure 5.33: The influence of the threshold parameter in the hacked version of the Smart Camera scenario for the *Configuration Stability* for $M = L = 5$. The threshold ε is 0.1 (blue), 0.3 (red), and 0.5 (green). The duration of the attack is marked grey.

Figures 5.33, 5.34 and 5.35 show the influence of the three parameters in the *Configuration Stability* with suitable ranges in the hacked simulation. We see that, in this case, smaller values for the window size M and the lag parameter L are better. Higher values create wider spikes, shift the spikes further in the future, and, with $M = 20$, the combination with higher values of ε generates noisier graphs than with $M = 10$ or $M = 5$.

The parameter ε influences the number of spikes. With $M = L = 10$, we get three spikes for $\varepsilon = 0.5$, five for $\varepsilon = 0.3$ and ten spikes for $\varepsilon = 0.1$. This behaviour (with different number of spikes) is the same with the other window sizes.

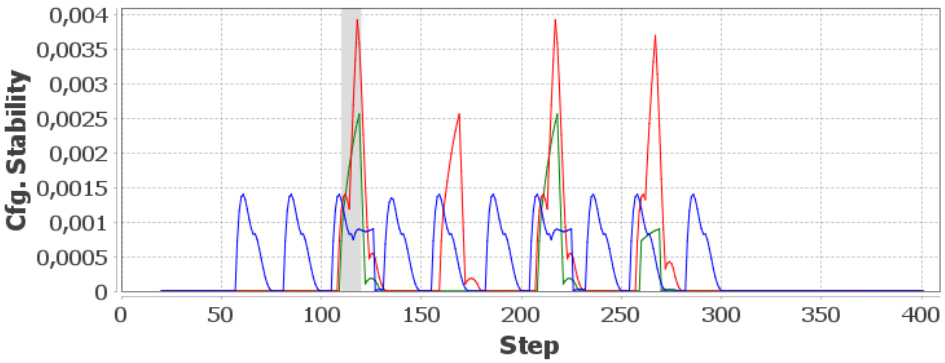


Figure 5.34: The influence of the threshold parameter in the hacked version of the Smart Camera scenario for the *Configuration Stability* for different $M = L = 10$. The threshold ε is 0.1 (blue), 0.3 (red), and 0.5 (green). The duration of the attack is marked grey.

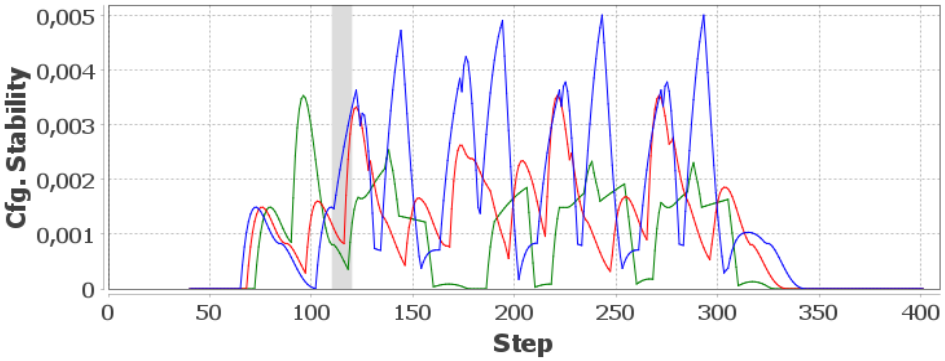


Figure 5.35: The influence of the threshold parameter in the hacked version of the Smart Camera scenario for the *Configuration Stability* for different $M = L = 20$. The threshold ε is 0.1 (blue), 0.3 (red), and 0.5 (green). The duration of the attack is marked grey.

Table 5.5 gives an overview of the values of the first spike after $t = 100$ when the given parameters are applied. This spike is chosen only because it allows a good comparison of the influence of the parameters.

Figure 5.36 compares the two simulations for $M = L = 5$. The Configuration Stability gives identical values until $t = 152$, more than 30 time steps after the attack ended. Beginning with this point, the influence of the attack becomes visible. In the undisturbed case, the metric reaches its third peak of $c_s \approx 0.006$ at $t = 158$. The hacked simulation gives a corresponding peak of $c_s \approx 0.0025$ at $t = 159$. For the fourth and the fifth spike,

M	L	ε	**Value**	**Location**	**Figure**
5	5	0.1	0.0023	$t = 109$	5.33 blue
5	5	0.3	0.0036	$t = 109$	5.33 red
5	5	0.5	0.0016	$t = 110$	5.33 green
10	10	0.1	0.0014	$t = 109$	5.34 blue
10	10	0.3	0.0039	$t = 118$	5.34 red
10	10	0.5	0.0026	$t = 119$	5.34 green
20	20	0.1	0.0036	$t = 122$	5.35 blue
20	20	0.3	0.0033	$t = 122$	5.35 red
20	20	0.5	0.0025	$t = 136$	5.35 green

Table 5.5: The influence of the tuning parameters for the *Configuration Stability* on the location and height of the peaks after the disturbance in the Smart Camera scenario. The values are for the first spike after $t = 110$.

the values are switched.

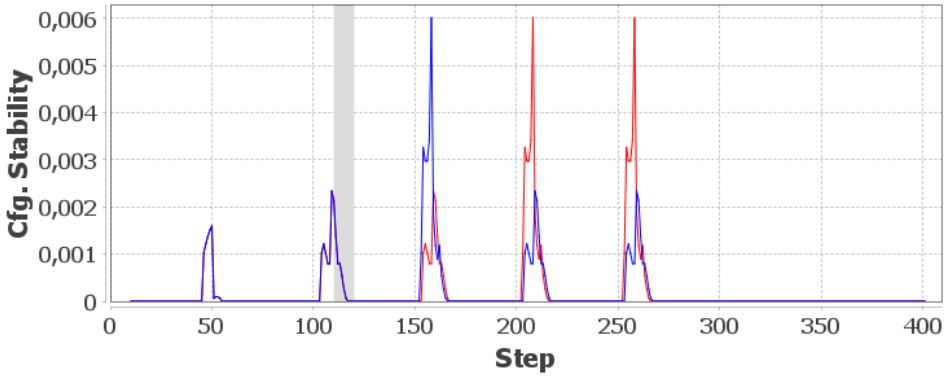


Figure 5.36: The time series of the *Configuration Stability* for the Smart Camera scenario with $M = L = 5$ and $\varepsilon = 0.1$. Blue is the undisturbed simulation, and red shows the hacked simulation. The duration of the attack is marked grey.

Configuration Variability

Figure 5.37 presents the *Configuration Variability* of the two simulations. Here, the attack shows immediate effects in the graph when compared to the expected/undisturbed behaviour. First, an unexpected spike of $c_v \approx 0.31$ is created at $t = 112$, and after the attack, the baseline stays at $c_v \approx 0.7$ instead of the expected value of $c_v \approx 0.1$.

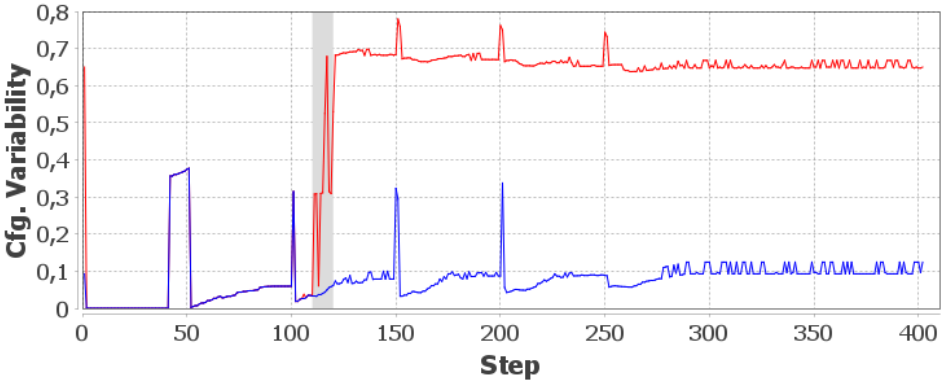


Figure 5.37: The time series of the *Configuration Variability* for the Smart Camera scenarios. Blue is the undisturbed simulation, and red shows the hacked simulation. The duration of the attack is marked grey.

Coherence Metrics

Figures 5.38 and 5.39 show the results for the *Coherence* metrics.

The attack on the camera angles shows no significant impact on the *Configuration Coherence*. Only after $t = 150$, the spikes differ in height, but less than 2.1%.

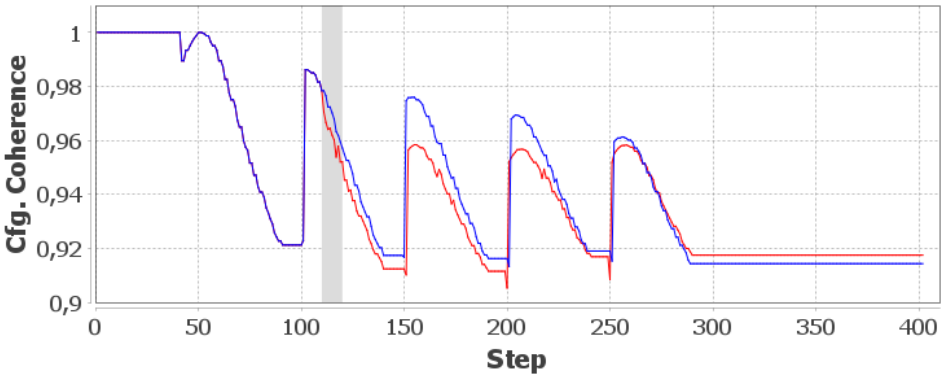


Figure 5.38: The time series of the *Configuration Coherence* for the Smart Camera scenario. Blue is the undisturbed simulation, and red shows the hacked simulation. The duration of the attack is marked grey.

The *Adaptation Coherence*, on the other hand, shows an additional peak of $coh_a \approx 0.89$ at $t = 112$ and subsequently no spikes where the undisturbed simulation suggests it. The four spikes between $t = 150$ and $t = 220$ are missing in the disturbed simulation, as well as the one at $t = 252$. Taken alone, the *Adaptation Coherence* cannot identify the attack. Although a peak is created, this peak is not isolated enough when compared to the

height of the expected ones. Only when directly compared to the expected behaviour does the attack become visible due to the point in time where the peak occurs unexpectedly.

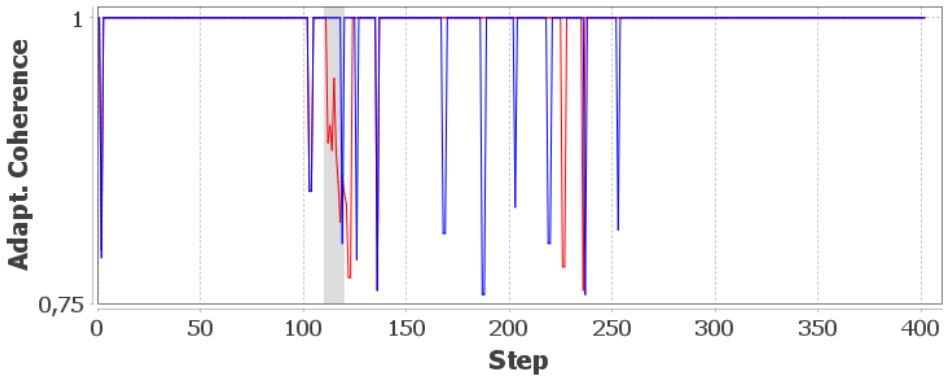


Figure 5.39: The time series of the *Adaptation Coherence* for the Smart Camera scenario. Blue is the undisturbed simulation, and red shows the hacked simulation. The duration of the attack is marked grey.

Parameter Usage Metrics

The influence of different window sizes in the Parameter Usage metrics can be seen in Figures 5.40 and 5.41. While the *Global Parameter Usage* reacts delayed (with a delay of M steps) on the camera handover, the *Average Parameter Usage* shows an immediate effect that, in turn, lasts longer than the effect in the *Global Parameter Usage*.

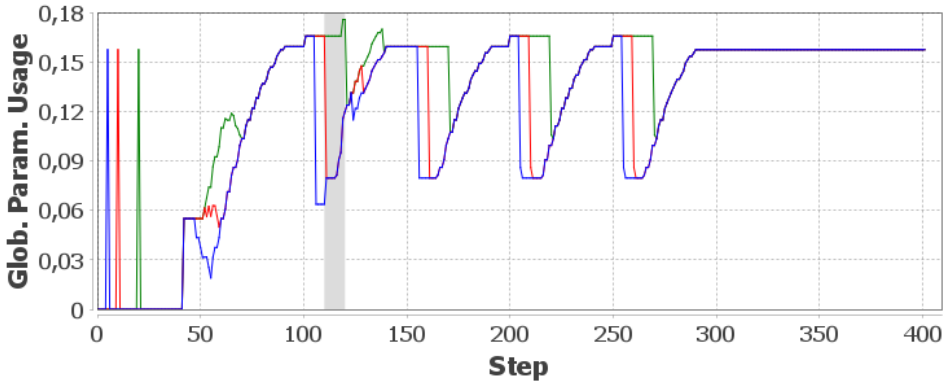


Figure 5.40: The time series of the *Global Parameter Usage* for the hacked simulation of the Smart Camera scenario with different window sizes. Blue: $M = 5$, red: $M = 10$, green: $M = 20$. The duration of the attack is marked grey.

Furthermore, we see that for $M = 5$ and $M = 10$, the height of the peaks does not differ for both metrics, except the ones during the attack. For example, the dips reach

a value of $U_g \approx 0.08$ for the *Global Parameter Usage*. For $M = 20$, they drop only to $U_g \approx 0.1$.

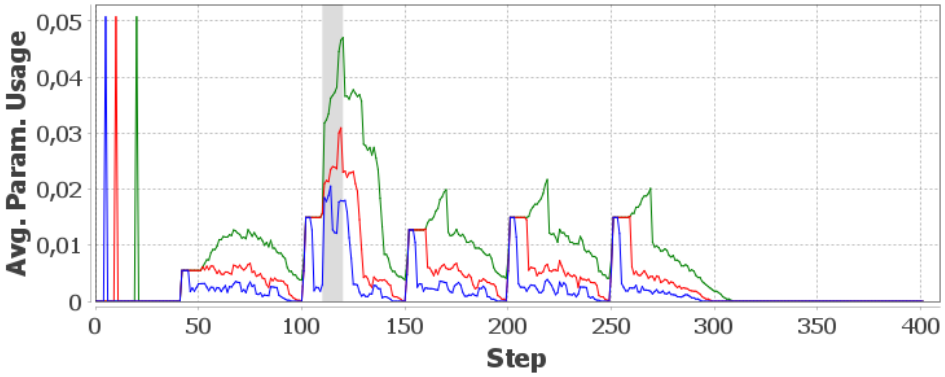


Figure 5.41: The time series of the *Average Parameter Usage* for the hacked simulation of the Smart Camera scenario with different window sizes. Blue: $M = 5$, red: $M = 10$, green: $M = 20$. The duration of the attack is marked grey.

Figures 5.42 and Figures 5.43 show the comparison of the hacked simulation with the undisturbed case.

The *Global Parameter Usage* shows only small differences between the graphs. During the attack, a slightly advanced rise at $t = 111$ and subsequently less deep dips of $U_g \approx 0.8$ are notable. The undisturbed case drops to $U_g \approx 0.63$ in the second and third dip. Taken alone, these differences are not strong enough that the metric could identify the attack.

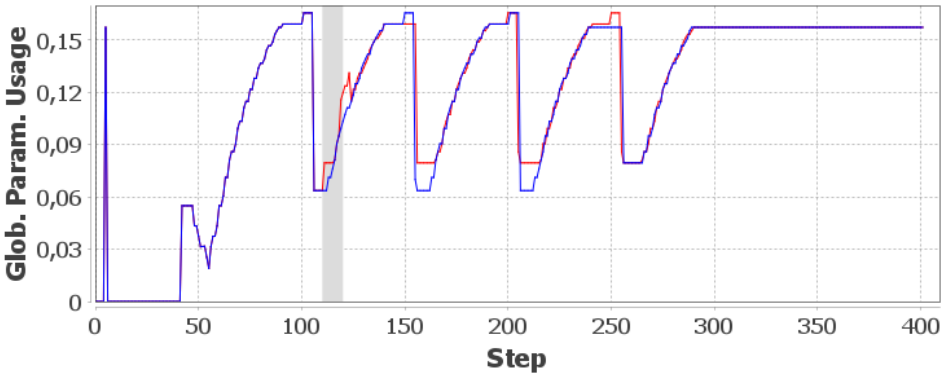


Figure 5.42: The time series of the *Global Parameter Usage* for the hacked (red) and the undisturbed simulation (blue) with $M = 5$. The duration of the attack is marked grey.

The *Average Parameter Usage*, on the other hand, reacts with an immediate and steep increase during the attack that lasts for the time of the attack. It rises to $U_a \approx 0.018$ at $t = 111$, while it stays at the lower baseline of $U_a \leq 0.004$ in the undisturbed case.

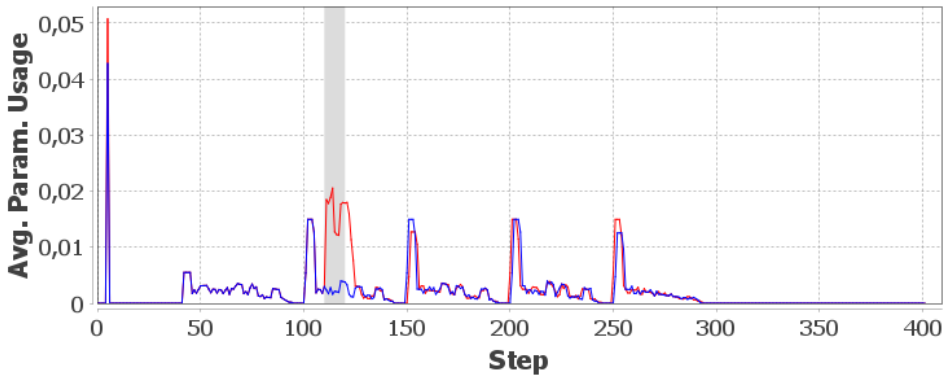


Figure 5.43: The time series of the *Average Parameter Usage* for the hacked (red) and the undisturbed simulation (blue) with $M = 5$. The duration of the attack is marked grey.

Configuration Divergence

In Figures 5.44, 5.45 and 5.46, we see the *Configuration Divergence* with different parameter values. All presented combinations are able to identify the attack when compared to the expected time series by the unexpected peak at $t = 110 + M$. Furthermore, we see that higher values for the window parameter M lead to a more delayed reaction of the graph. The corresponding peak appears with a delay of M steps.

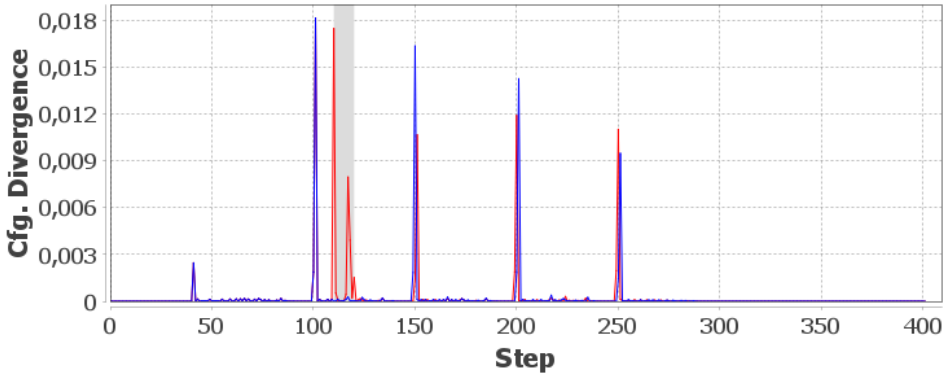


Figure 5.44: The *Configuration Divergence* in the Smart Camera scenario for $M = L = 1$ in the hacked (red) and the undisturbed simulation (blue). The duration of the attack is marked grey.

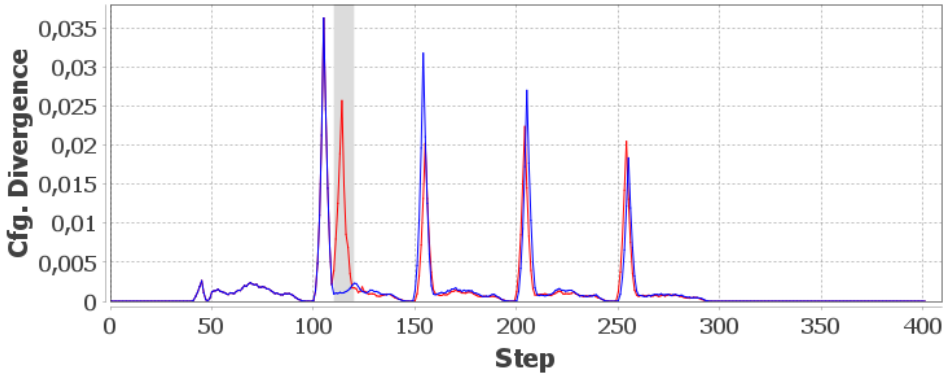


Figure 5.45: The *Configuration Divergence* in the Smart Camera scenario for $M = L = 5$ in the hacked (red) and the undisturbed simulation (blue). The duration of the attack is marked grey.

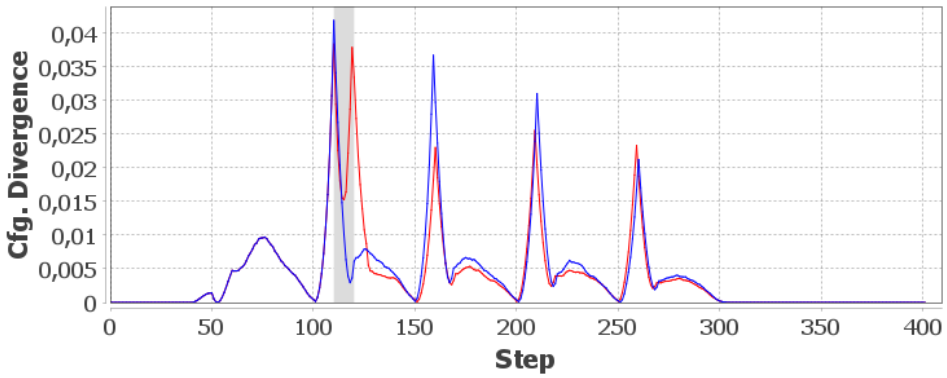


Figure 5.46: The *Configuration Divergence* in the Smart Camera scenario for $M = L = 10$ in the hacked (red) and the undisturbed simulation (blue). The duration of the attack is marked grey.

Self-Organisation Divergence

Again, as in the Flocking scenario, the cameras do not communicate directly, but they broadcast their current configuration. Under this assumption the *Self-Organisation Divergence* is, again, equal to the *Configuration Divergence*.

Summary

In this scenario, the *Configuration Coherence* and the *Global Parameter Usage* are not able to identify the attack. Their time series differ only slightly from the expected values. The *Configuration Stability* reacts visibly but with a high delay. With a small window size and a small delay value, the *Configuration Divergence* reacts immediately, as do the

remaining metrics. Yet, only the *Configuration Variability* shows a unique change in the values so that the attack is visible without the comparison to the expected results.

5.2.4 Results for the Artificial Road Block Scenario

Here, we investigate the results of the Artificial Road Block scenario. The points of interest are the start of the road block at $t = 250$ and its end at $t = 400$. As explained in the presentation of the scenario (see Chapter 5.1.3), we will consider each signal light as an individual subsystem.

Entropy

Figures 5.47, 5.48 and 5.49 show the *Entropy* for different estimator bin counts. During the simulation, the intersections take configuration values in the interval $[0, 40]$. Therefore, we use fractions of 40 as bin counts. We see that smaller bins are better for identifying the start of the road block, while bigger bins are better for detecting its end.

For $M = 20$, both events are visible due to the change of the baseline in the according phases of the simulation. Before the beginning of the road block, from $t = 50$ to $t = 253$, the metric stays near $H \approx 1.05$. Then, it rises to $H \approx 2.0$ until $= 405$, where it drops to the third stable baseline at $H \approx 1.8$. However, the effects of the second event are not separable from other peaks, while the first event shows by the steep rise of the time series.

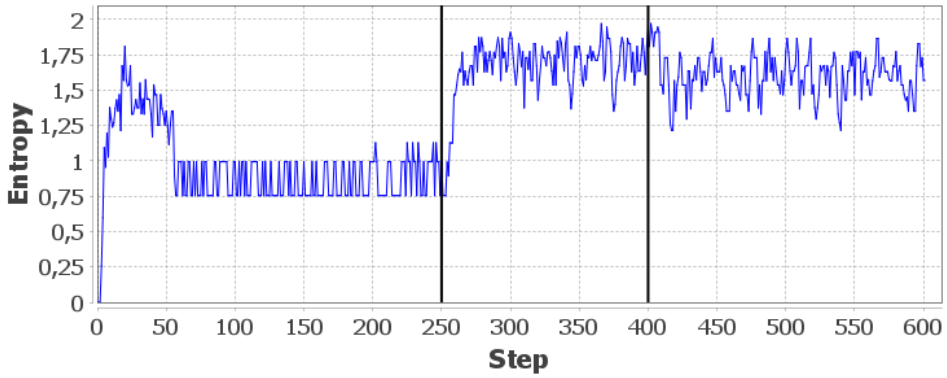


Figure 5.47: The time series of the *Entropy* for the Road Block scenario with bin count $M = 10$. The black lines denote the start and the end of the road block.

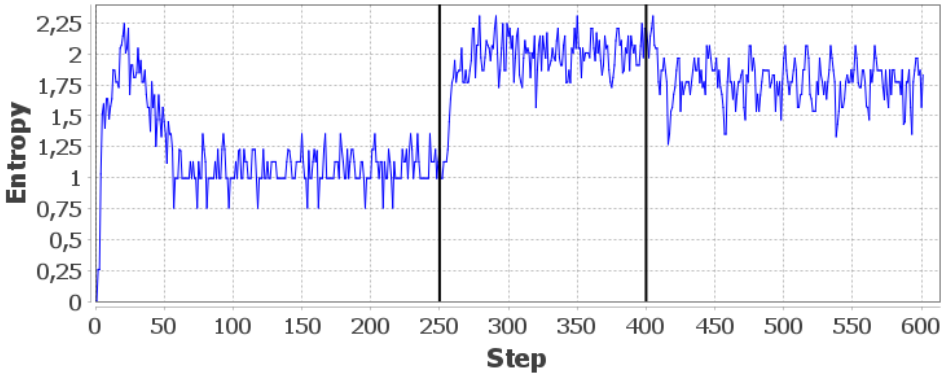


Figure 5.48: The time series of the *Entropy* for the Road Block scenario with bin count $M = 20$. The black lines denote the start and the end of the road block.

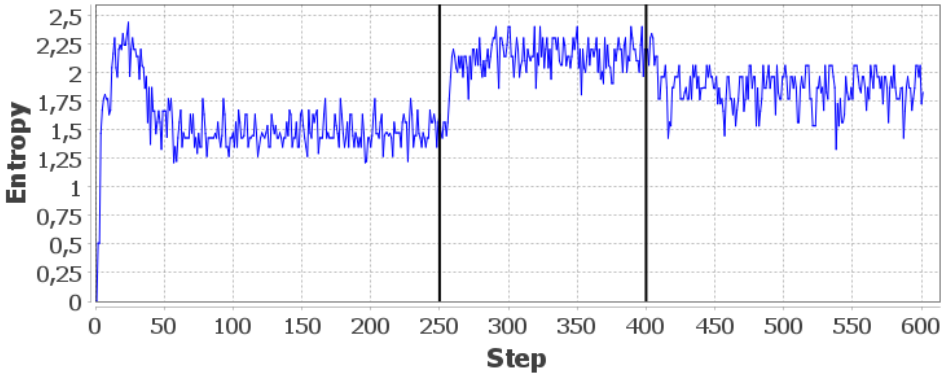


Figure 5.49: The time series of the *Entropy* for the Road Block scenario with bin count $M = 40$. The black lines denote the start and the end of the road block.

Configuration Stability

First, we start with an overview of the influence of the tuning parameters on the *Configuration Stability*. The metric creates a noisy time series for all parameter combinations in this scenario. Therefore, Figures 5.50, 5.51 and 5.52 show a detailed view of the time around the end of the blockade at $t = 400$.

We see that $M = L = 5$ produces a very noisy time series where the impact of the disturbance does not show at all. The removal of the road block has no impact in this case.

For $M = L = 10$, the removal becomes visible only for $\varepsilon = 5$ with $c_s \approx 0.03$ at $t = 425$ and $\varepsilon = 10$ with $c_s \approx 0.025$ at $t = 423$. On the other hand, $\varepsilon = 5$ creates higher spikes at uninteresting time steps, e.g. at $t = 412$ which is not of interest due to the delay of the metric.

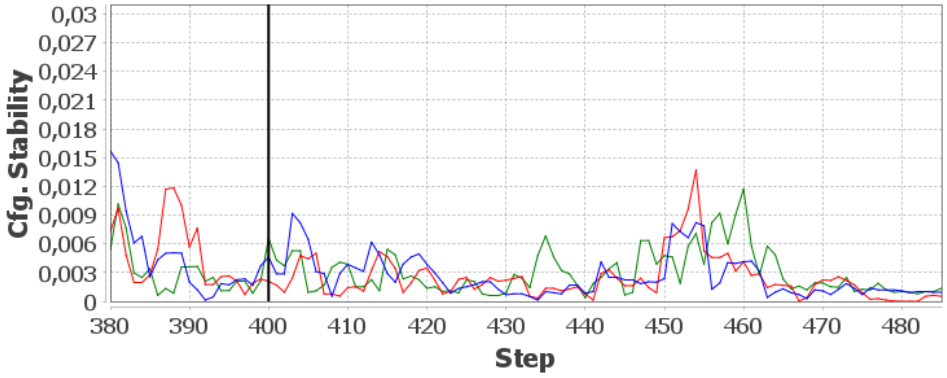


Figure 5.50: The influence of the threshold parameter in the Road Block scenario for the *Configuration Stability* for $M = L = 5$. Shown is the time frame around the end of the blockade at $t = 400$ (black line). The threshold ε is 5 (blue), 10 (red), and 20 (green).

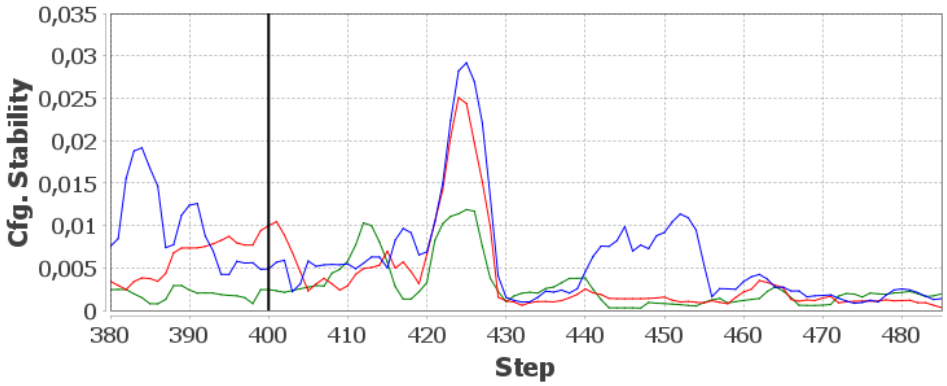


Figure 5.51: The influence of the threshold parameter in the Road Block scenario for the *Configuration Stability* for $M = L = 10$. Shown is the time frame around the end of the blockade at $t = 400$ (black line). The threshold ε is 5 (blue), 10 (red), and 20 (green).

With $M = L = 20$, bigger thresholds are needed to detect the removal of the road block. For $\varepsilon = 20$, the Configuration Stability gives a value of $c_s \approx 0.035$ at $t = 446$, and $\varepsilon = 20$ leads to $c_s \approx 0.032$ at $t = 443$. $\varepsilon = 5$ creates a peak of $c_s \approx 0.021$ at $t = 446$ which is below the uninteresting peak at $t = 415$.

By choosing $M = L = \varepsilon = 10$, we get Figure 5.53. Here, we see that after the initial setup phase, the time series has a constant value of $c_v = 0$ until $t = 255$. Then, the impact of the road block becomes visible with the two peaks at $t = 262$ and $t = 271$, where the time series reaches $c_s \approx 0.03$. After the start of the road block, intersections need to adapt very often, which leads to an unstable graph. The end of the road block is visible in the peak $t = 424$ with $c_s \approx 0.025$. The simulation then remains in a state where adaptations take place, but not as strong as during the road block.

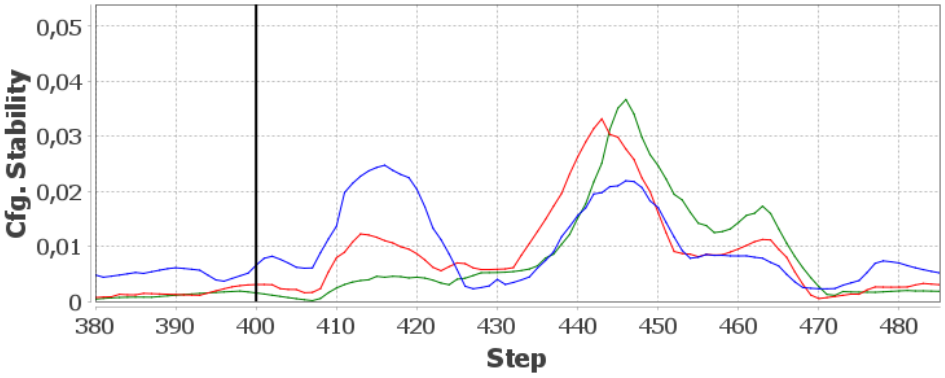


Figure 5.52: The influence of the threshold parameter in the Road Block scenario for the *Configuration Stability* for $M = L = 20$. Shown is the time frame around the end of the blockade at $t = 400$ (black line). The threshold ε is 5 (blue), 10 (red), and 20 (green).

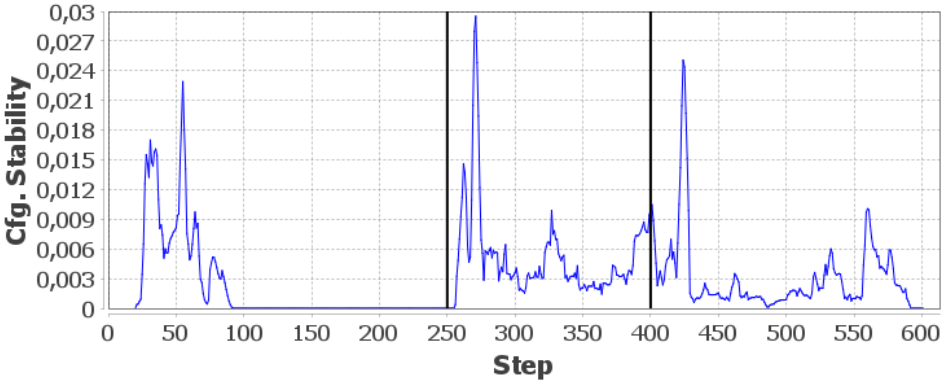


Figure 5.53: The time series of the *Configuration Stability* for the Road Block scenario with $M = L = \varepsilon = 10$. The black lines denote the start and the end of the road block.

Configuration Variability

In Figure 5.54, we see the time series of the *Configuration Variability*. Again, we can distinguish four phases: the setup, the stable phase, the phase between the start and the end of the road block, and the time after the road block. During the latter, we see a more volatile behaviour than during the road block. The last two phases cannot be clearly distinguished. The end of the road block leads to a drop in the graph from $c_v \approx 7.2$ to $c_v \approx 5.3$ at $t = 410$. However, it is distinguishable from other peaks in the graph that show comparable amplitudes. The start of the road block, on the other hand, can be easily identified. At $t = 256$ the graph rises suddenly from $c_v \approx 3.4$ to the new baseline at $c_v \approx 6.0$.

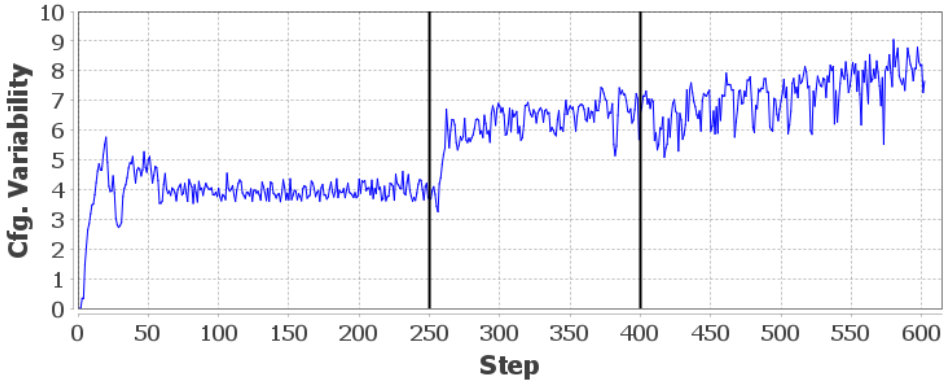


Figure 5.54: The time series of the *Configuration Variability* for the Road Block scenario. The black lines denote the start and the end of the road block.

Coherence Metrics

Figure 5.55 shows the result for the *Configuration Coherence*. Since the graph appears relatively flat when including all values, we additionally plotted a detailed view the time series in Figure 5.56. Again, the initial setup is followed by a phase of higher coherence. This aligns with the behaviour of the *Configuration stability*. With the beginning of the road block, both time series show a decreased coherence. At $t = 257$, the *Configuration Coherence* leaves its previous baseline of $coh_c \approx 0.008$ to its new baseline at $coh_c \approx 0.004$, where it stays for the rest of the simulation.

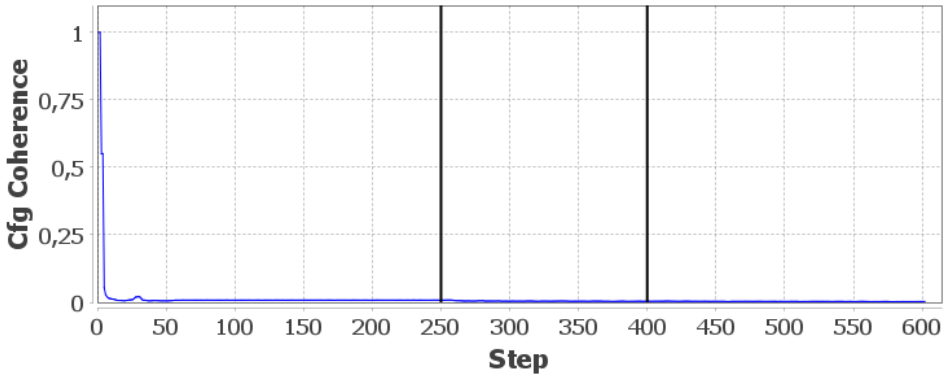


Figure 5.55: The time series of the *Configuration Coherence* for the Road Block scenario. The black lines denote the start and the end of the road block.

Similarly, the *Adaption Coherence* (Figure 5.57) never drops below $coh_a = 0.2$ during the phase before the road block. With the beginning of the road block again at $t = 257$, it drops to a baseline of $coh_a \approx 0.17$ and stays there. Notably, the end of the road block

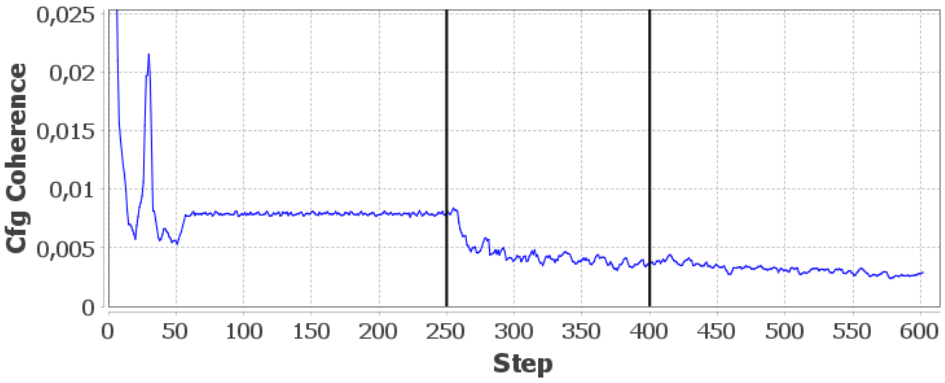


Figure 5.56: Detailed graph of the time series of the *Configuration Coherence* for the Road Block scenario. The black lines denote the start and the end of the road block.

is not visible in the graphs.

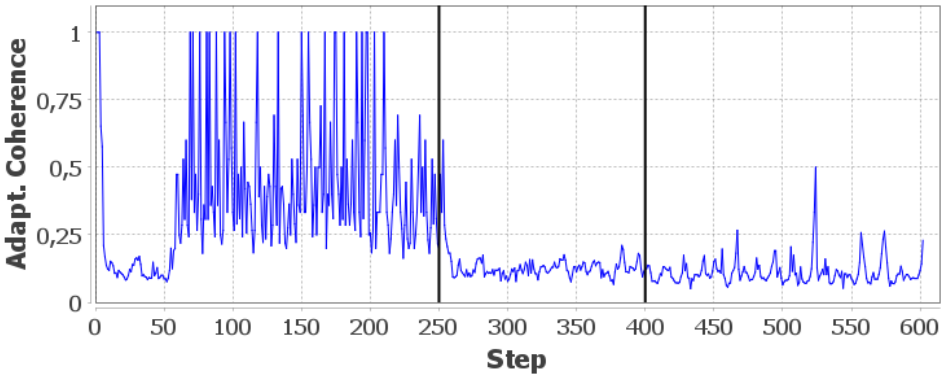


Figure 5.57: The time series of the *Adaptation Coherence* for the Road Block scenario. The black lines denote the start and the end of the road block.

Parameter Usage Metrics

Figure 5.58 gives the *Global Parameter Usage* with different window sizes. We see that with the beginning of the road block, the intersections start to use a wider range of configuration values without reverting back to smaller intervals. This leads to increasing values of the time series. Except during the setup phase, the window size M has no influence on the time series. After $t = 71$, the graphs are identical for all given parameters. The stable phase and the start of the road block are again clearly visible in the rise from $U_g \approx 0.55$ at $t = 258$ to $U_g \approx 0.72$ at $t = 262$. The end of the road block, on the other hand, shows no influence on this metric.

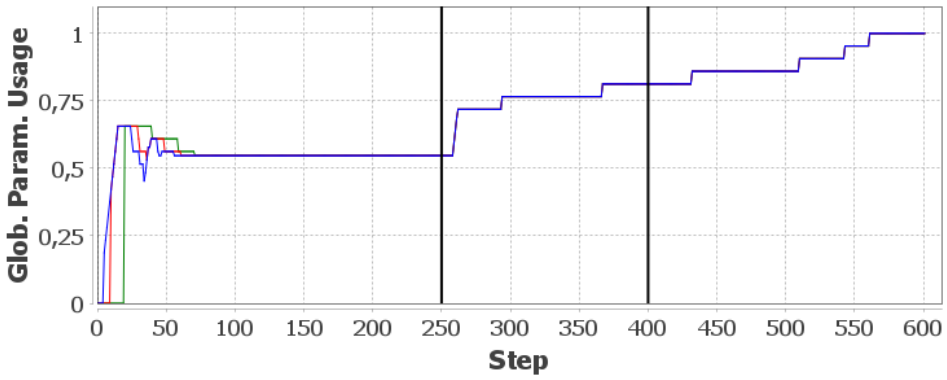


Figure 5.58: The time series of the Global Parameter Usage for the Road Block scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The black lines denote the start and the end of the road block.

In Figure 5.59, we see the *Average Parameter Usage* with different window sizes. Again, longer time windows lead to longer influences of events on the graph. With all three window sizes, the metric shows the beginning of the disturbance at $t = 253$, where the metric leaves its previous baseline of $U_a \approx 0.15$ to a higher baseline. For example, with $M = 10$ this new baseline is at $U_a \approx 1$. In contrast to the *Global Parameter Usage*, this metric is able to identify the end of the road block but requires longer time windows for that. With $M = 20$, the metric creates an isolated peak of $U_a \approx 2$ at $t = 418$. For $M = 5$, this peak has the same height as several other peaks during the road block.

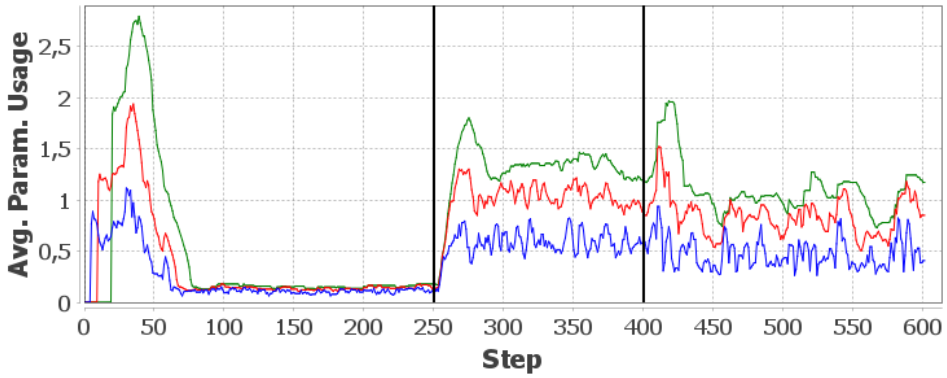


Figure 5.59: The time series of the Average Parameter Usage for the Road Block scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The black lines denote the start and the end of the road block.

Configuration Divergence

As we can see in Figures 5.60, 5.61 and 5.62, the tuning parameters of *Configuration Divergence* have a strong influence on the noisiness of the graph. Smaller window sizes M lead to a faster reaction, but bigger values in combination with a bigger lag value L lead to smoother graphs and better isolation of the beginning of the incident. For $M = L = 1$, we can identify an increase of the baseline from $c_d \approx 0.5$ to $c_d \approx 2$ with the beginning of the road block at $t = 255$ and a drop of the baseline after $t = 400$ to $c_d \approx 1.0$ for several steps and $c_d \approx 1.5$ after $t = 440$. Higher values for the parameters do not allow such an observation of strongly changing baselines. However, $M = L = 5$ leads to peaks that indicate the start and the end of the road block. The first peak is $c_d \approx 8.0$ at $t = 262$, and the second is $c_d \approx 8.25$ at $t = 462$. With $M = L = 10$, this second peak is lost again. Here, only the beginning of the road block shows with a peak of $c_d \approx 12.5$ at $t = 267$.

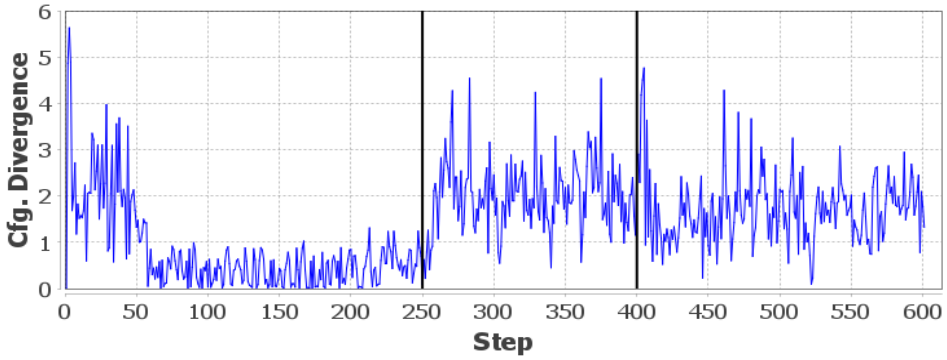


Figure 5.60: The time series of the *Configuration Divergence* for the Road Block scenario with $M = L = 1$. The black lines denote the start and the end of the road block.

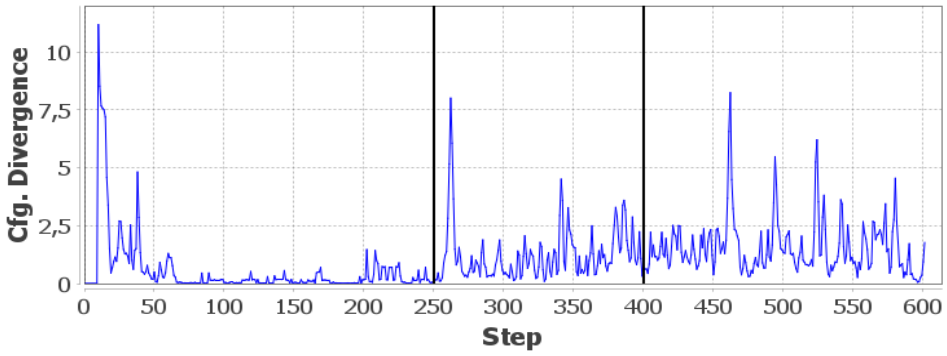


Figure 5.61: The time series of the *Configuration Divergence* for the Road Block scenario with $M = L = 5$. The black lines denote the start and the end of the road block.

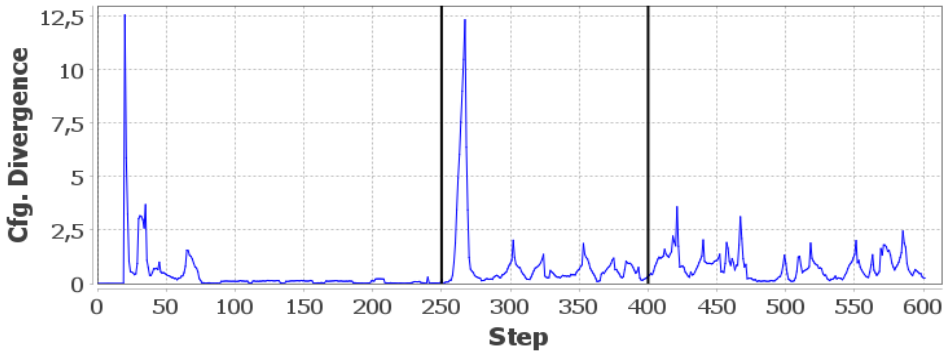


Figure 5.62: The time series of the *Configuration Divergence* for the Road Block scenario with $M = L = 10$. The black lines denote the start and the end of the road block.

Self-Organisation Divergence

This scenario is implemented without any communication between the subsystems, and there is no mechanism for a subsystem to gain knowledge about the state of another one. Therefore, the *Self-Organisation Divergence* is not applicable in this scenario.

Summary

The simulation can be divided into four phases: setup (1), stable phase (2), the road block (3), and the phase after the road block (4). A clear distinction between the phases 3 and 4 is only given by the *Entropy*. The beginning of the road block has a notable impact on all time series, but only the *Configuration Stability* and the *Average Parameter Usage* clearly indicate the transition from phase 2 to 3 and from 3 to 4 with their notable underlying events with isolated peaks.

5.2.5 Results for the Artificial Rush Hour Scenario

With the same arguments (see Chapter 5.1.3) as for the Artificial Road Block scenario, we will consider each signal light as a single subsystem. In this section, we will analyse the behaviour of the biggest disturbance, where 500 additional cars spawn during the rush hours. The two other simulations with 300 and 100 extra cars are presented in Chapter 5.2.6. In this scenario, our attention is on the start of the two rush hour phases at $t = 250$ and $t = 750$.

Entropy

Figure 5.63 shows the *Entropy* for different estimator bin counts. For the same reasons as in the Rush Hour scenario, we use fractions of 40 as bin counts. In contrast to the

previous scenarios, in this case, the bin count relates to the baseline of the graph. The bigger the bin count, the higher the value of the *Entropy*. In this scenario, we see that the bin count with the given values has no significant influence on the ability to detect the rush hour events. The first rush hour creates no distinguishable change in the time series. The second rush hour leads to a notable spike at $t = 769$. With $M = 20$, this peak reaches a value of $H \approx 2.21$, while the baseline is at $H \approx 1.85$.

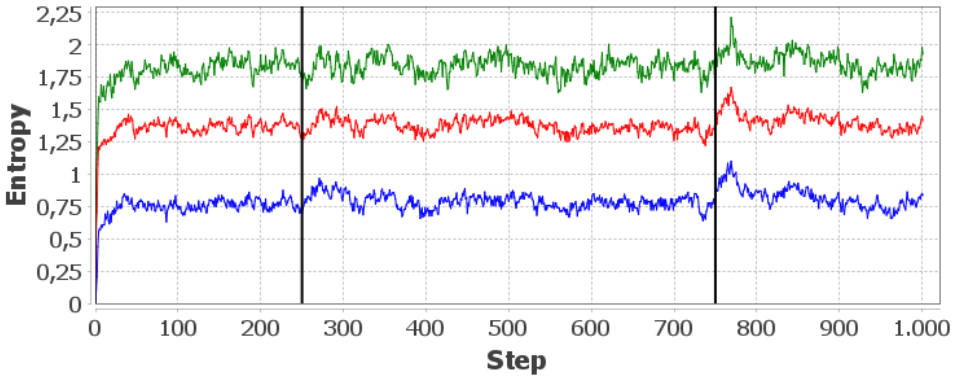


Figure 5.63: The time series of the *Entropy* for the Rush Hour scenario with bin counts $M = 10$ (blue), $M = 20$ (red), and $M = 40$ (green). The black lines denote the start of the two rush hours.

Configuration Stability

Again, we start with the overview of the parameters for the *Configuration Stability*. For a better overview of the parameters' influence, Figures 5.64, 5.65 and 5.66 show the time frame around the second rush hour. We see that smaller values for the threshold ε create more peaks. Another observation is that the peaks resulting from the rush hour have nearly the same height for $\varepsilon = 10$ and $\varepsilon = 20$. For example, both create a peak of $c_s \approx 0.0015$ at $t = 783$. For the different windows sizes, the height of corresponding peak for $\varepsilon = 5$ differs notably from the height of the other two. Again, higher values for M and L lead to a delay of the peak. With $M = L = 20$, the peaks for the second rush hour is at $t = 790$, while for $M = L = 10$, they are eight steps earlier.

With $M = L = 10$ and $\varepsilon = 20$, we get Figure 5.67. Both rush hours create isolated peaks in the graph. The first rush hour gives a value of $c_s \approx 0.0007$ at $t = 266$. We see that during the second rush hour, the effect of the adaptation is delayed. This peak appears at $t = 782$ and reaches $c_s \approx 0.0022$.

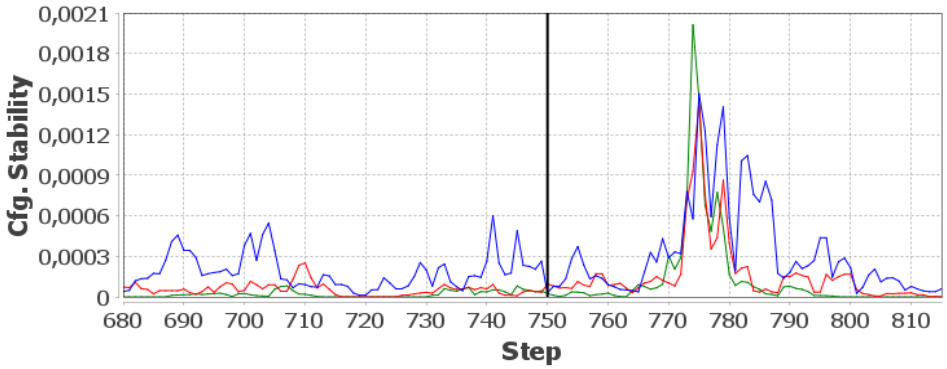


Figure 5.64: The influence of the threshold parameter in the Rush Hour scenario for the *Configuration Stability* for $M = L = 5$. The threshold ε is 5 (blue), 10 (red), and 20 (green). The black line denotes the start of the second rush hour.

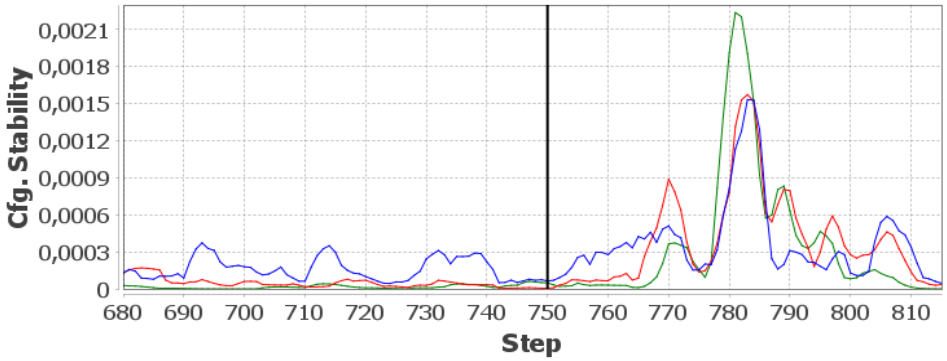


Figure 5.65: The influence of the threshold parameter in the Rush Hour scenario for the *Configuration Stability* for $M = L = 10$. The threshold ε is 5 (blue), 10 (red), and 20 (green). The black line denotes the start of the second rush hour.

Configuration Variability

In Figure 5.68, we see the time series of the *Configuration Variability*. The metric creates isolated peaks as a result of the rush hours. The first peak is at $t = 266$ with $c_v \approx 2.4$, and the second is at $t = 763$ with $c_v \approx 2.37$. The phases after both rush hours give an increased baseline with a value of $c_v \approx 1.5$ that deviates notably from the baseline ($c_v \approx 1.25$).

Coherence Metrics

Figures 5.69 and 5.70 show the result for the *Coherence* metrics. We see that both time series react with isolated peaks on both rush hour events.

The *Configuration Coherence* drops to $coh_c \approx 0.015$ at $t = 265$ as a reaction to

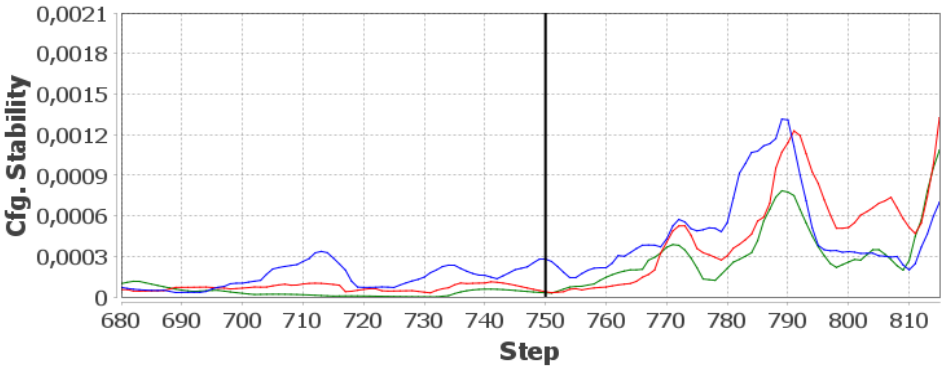


Figure 5.66: The influence of the threshold parameter in the Rush Hour scenario for the *Configuration Stability* for $M = L = 20$. The threshold ε is 5 (blue), 10 (red), and 20 (green). The black line denotes the start of the second rush hour.

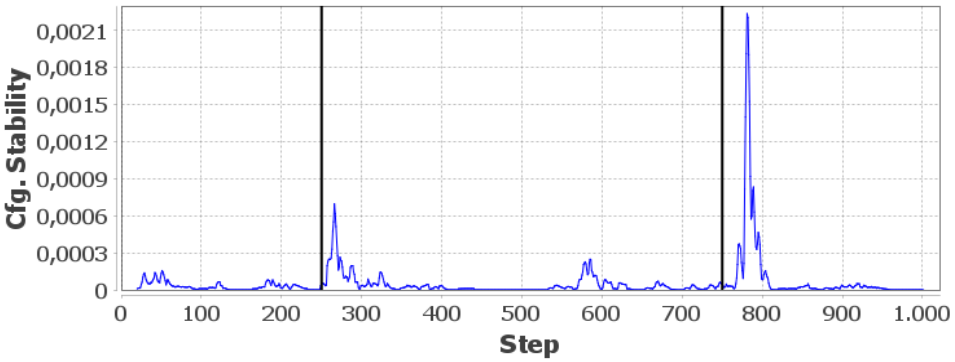


Figure 5.67: The time series of *Configuration Stability* for the Rush Hour scenario with $M = L = 10$ and $\varepsilon = 20$. The black lines denote the start of the two rush hours.

the first event. The second event shows as another drop to $coh_c \approx 0.015$ at $t = 762$. Furthermore, the *Configuration Coherence* shows that the simulation needs about 100 time steps after the disturbance with a coherence value near $coh_c \approx 0.03$ to reach the same coherence level (value $coh_c > 0.04$) as before.

Although the *Adaptation Coherence* shows a noisier behaviour with a baseline of $coh_c \approx 0.3$, the two drops are present here, too. The first reaches a value of $coh_a \approx 0.10$ at $t = 762$, and the second is at $t = 766$ with $coh_c \approx 0.11$.

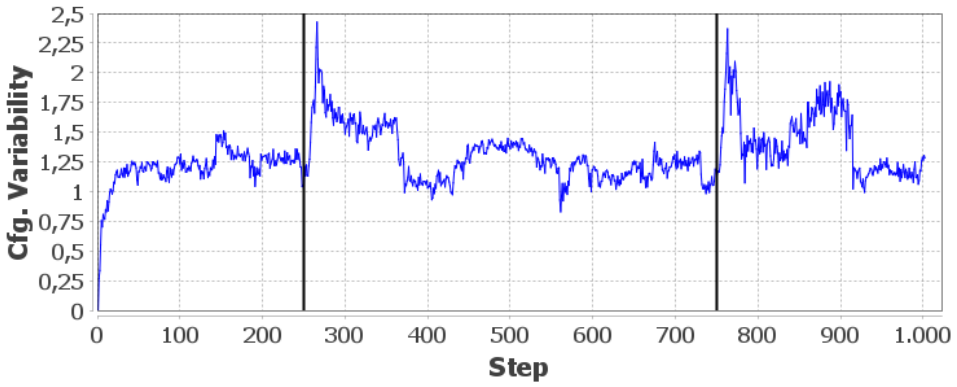


Figure 5.68: The time series of the *Configuration Variability* for the Rush Hour scenario. The black lines denote the start of the two rush hours.

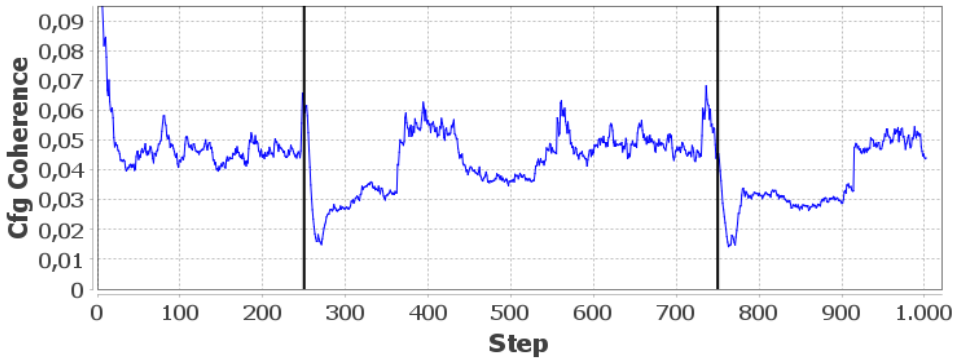


Figure 5.69: The time series of the *Configuration Coherence* for the Rush Hour scenario. Zoomed in for a better overview. The black lines denote the start of the two rush hours.

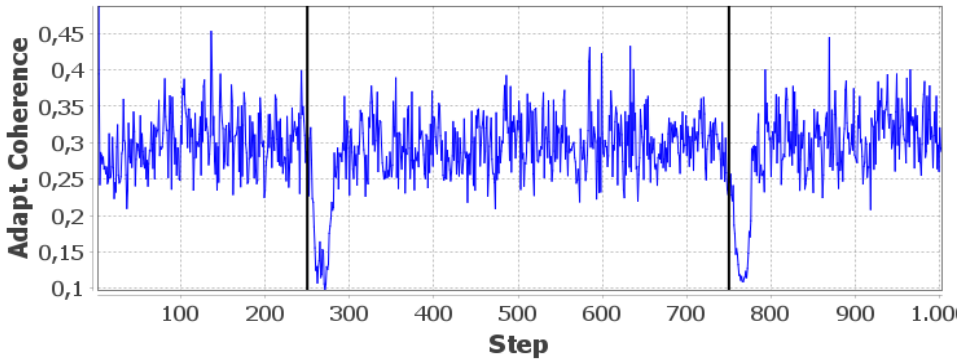


Figure 5.70: The time series of the *Adaptation Coherence* for the Rush Hour scenario. Zoomed in for a better overview. The black lines denote the start of the two rush hours.

Parameter Usage Metrics

Figure 5.71 shows the *Global Parameter Usage* with different window sizes. We see a baseline at $U_g \approx 0.38$. The first rush hour leads to a short phase, starting at $t = 268$, in which the system uses the complete range of available configuration values, as indicated by the value 1 for the metric. However, this phase is short enough to generate an isolated peak in the graph. The second rush hour creates a peak at $U_g \approx 0.82$ and is followed by a phase with values above the general baseline. The size of the time window only affects the duration of plateaus but not their value.

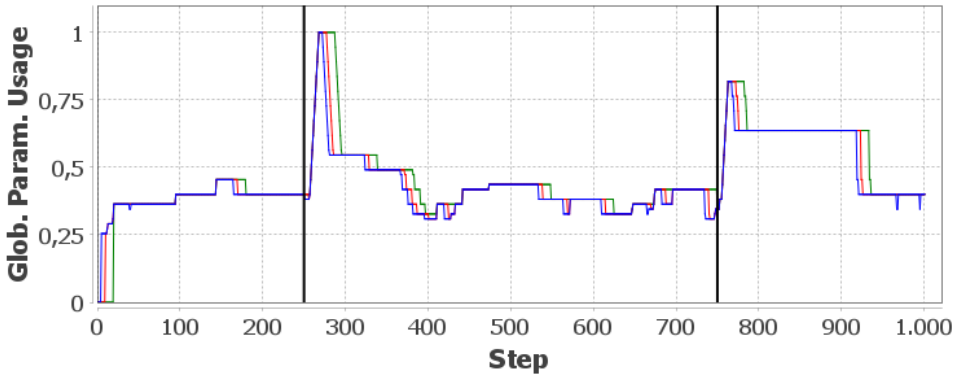


Figure 5.71: The time series of the *Global Parameter Usage* for the Rush Hour scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The black lines denote the start of the two rush hours.

In Figure 5.72, we see the *Average Parameter Usage* with different window sizes. The time series has a general baseline depending on the window size (e.g. $U_a \approx 0.16$ for $M = 5$. In contrast to the *Global Parameter Usage*, here it is the second rush hour that creates a higher peak than the first rush hour in all graphs. This peak reaches $U_a \approx 0.66$ at $t = 777$ for $M = 20$. For $M = 10$, this peak is at $t = 772$ with $U_a \approx 0.56$. With $M = 5$, we see the peak reaching $U_a \approx 0.35$ at $t = 768$.

For all time windows, the second rush hour remains identifiable, while the first one gets lost in the shuffle of peaks for smaller window sizes.

Configuration Divergence

Figures 5.73, 5.74 and 5.75 present the *Configuration Divergence* time series for different parameters. Again, higher values for the parameters lead to smoother graphs. The first peak is at $t = 267$, where it reaches $c_d \approx 1.42$ for $M = L = 1$. With $M = L = 10$, the metric gives a value of $c_d \approx 2.46$. The peak for the second rush hour is at $t = 766$.

Both events in the simulation create prominent peaks. However, a third event that

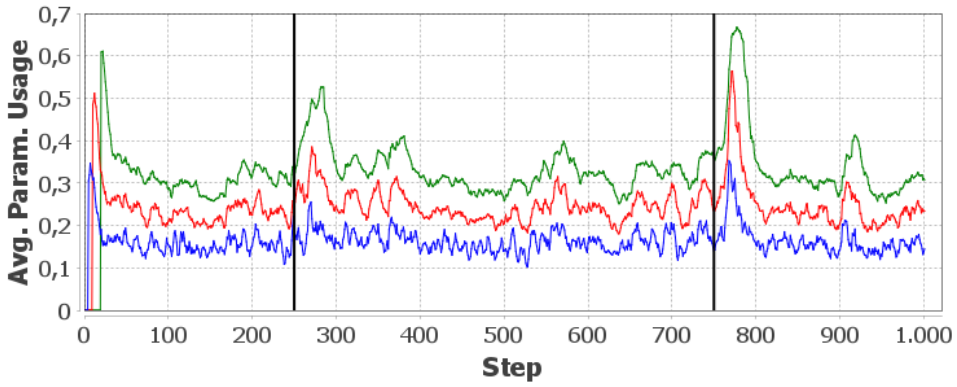


Figure 5.72: The time series of the *Average Parameter Usage* for the Rush Hour scenario with window sizes $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The black lines denote the start of the two rush hours.

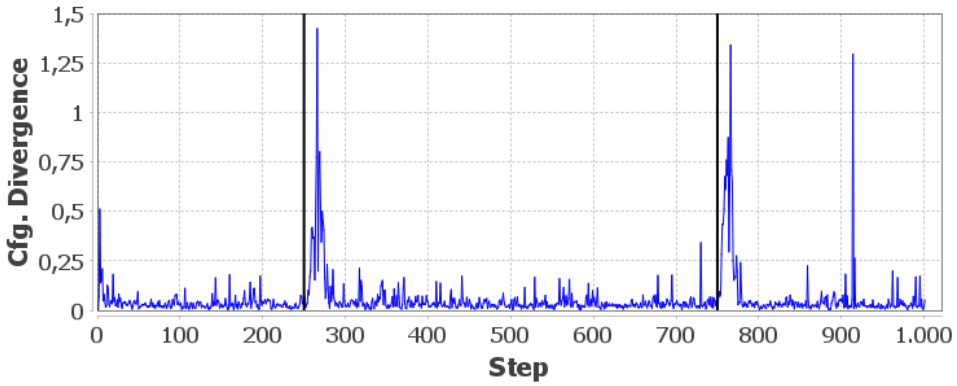


Figure 5.73: The time series of the *Configuration Divergence* for the Rush Hour scenario with $M = L = 1$. The black lines denote the start of the two rush hours.

is not related to the artificial disturbance creates another spike near $t = 900$.

Self-Organisation Divergence

This scenario is implemented without any communication between the subsystems, and there is no mechanism such that one subsystem gains knowledge about the state of another one. Therefore, the *Self-Organisation Divergence* is not applicable in this scenario.

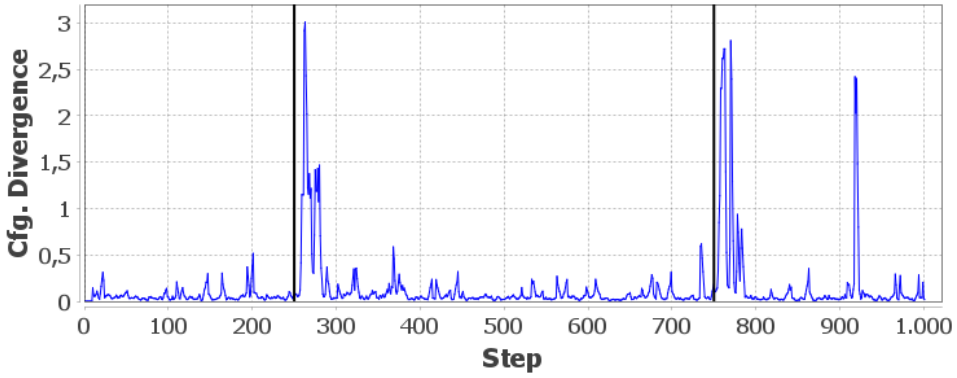


Figure 5.74: The time series of the *Configuration Divergence* for the Rush Hour scenario with $M = L = 5$. The black lines denote the start of the two rush hours.

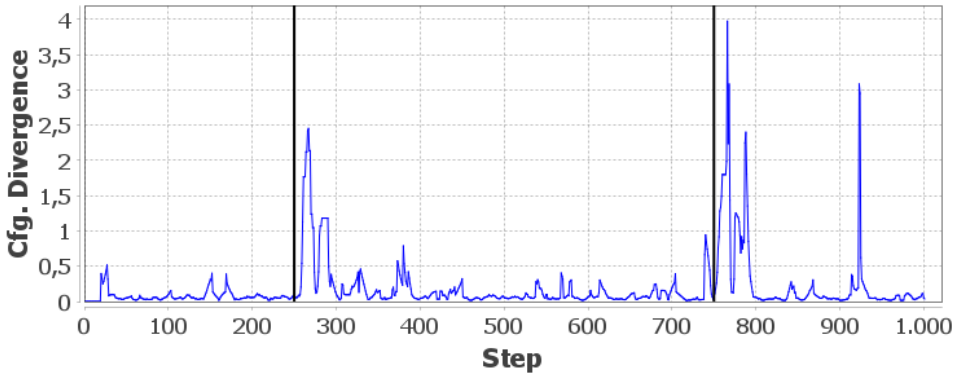


Figure 5.75: The time series of the *Configuration Divergence* for the Rush Hour scenario with $M = L = 10$. The black lines denote the start of the two rush hours.

Summary

All metrics are able to identify the two rush hour events, although most of them are more or less affected by the random behaviour of the simulation. The least noisy graphs are generated by the *Configuration Stability*, the *Global Parameter Usage*, and the *Configuration Divergence*. The latter shows a strong reaction to a third event that only the *Parameter Usage* metrics can identify clearly as well. The other metrics give no sharp indication for this point.

5.2.6 Results for the Artificial Rush Hour Scenario – Small Disturbances

In the same way as in Chapter 5.2.2, we will now evaluate how different strengths of the disturbance influence the metrics. For convenience, we will abbreviate the three disturbances with D_n, where n is the number of additional cars during the rush hours. Since we analysed D_{500} above, we will focus on the disturbances D_{300} and D_{100} in this section. The selected parameters for the metrics are, again, those which turned out suitable during the analysis of D_{500}. Again, the simulations run identically until the first rush hour appears. After that, the disturbances influence the dynamics in the simulation and lead to different results for the metrics. However, our attention stays on the rush hour periods at $t \geq 250$ and $t \geq 750$.

Entropy

Figures 5.76, 5.77 and 5.78 show the *Entropy* for the three disturbance sizes with bin count $M = 40$. The graphs are zoomed in to the period from shortly before the first rush hour to shortly after the second event. While the biggest disturbance D_{500} does not lead to a notable reaction for the first rush hour, D_{300} gives an isolated peak of $H \approx 2.09$ at $t = 271$. In D_{100}, on the other hand, it only rises to $H \approx 2.00$ at $t = 267$. This peak is not sufficiently prominent since this value is at the same level as the metric showed before the first rush hour.

The second rush hour leads to visible reactions in all three cases. In D_{500}, the peak reaches $H \approx 2.21$ at $t = 769$, which is about 10% above the previous upper bound. The respective peak in D_{300} is $H \approx 2.13$ at $t = 769$ and again sufficiently above the previous level. In D_{500}, the peak for the second rush hour is $H \approx 2.03$ at $t = 761$. Although it is at the same height that the *Entropy* reached during the first 250 steps, it is still an isolated peak due to the decreased maximum level of $H \approx 1.87$ that the metric took after the first rush hour.

However, the strength of the disturbance does not lead to a substantial difference in the reaction of the *Entropy* in this scenario.

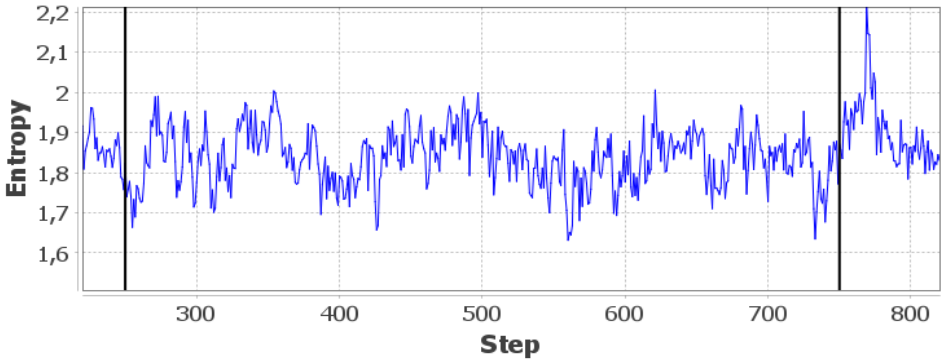


Figure 5.76: Detail view of the time series of the *Entropy* for the Rush Hour scenario with bin count $M = 40$ for the disturbance D_{500}. The black lines denote the start of the two rush hours.

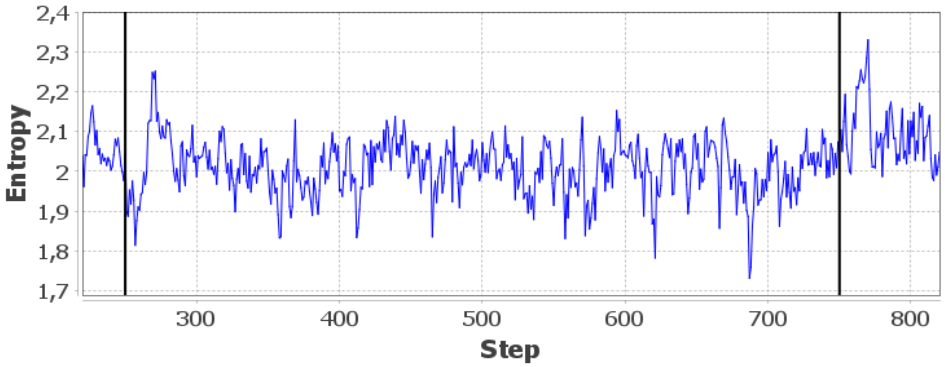


Figure 5.77: Detail view of the time series of the *Entropy* for the Rush Hour scenario with bin count $M = 40$ for the disturbance D_{300}. The black lines denote the start of the two rush hours.

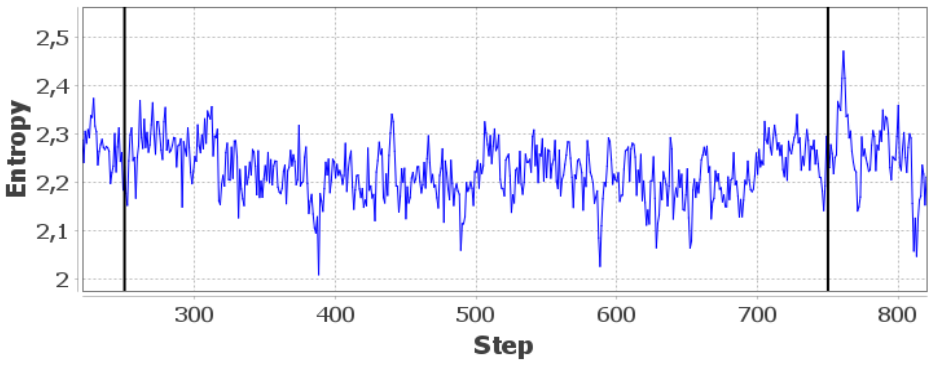


Figure 5.78: Detail view of the time series of the *Entropy* for the Rush Hour scenario with bin count $M = 40$ for the disturbance D_{100}. The black lines denote the start of the two rush hours.

Configuration Stability

Figure 5.79 shows the result of the *Configuration Stability* in the two minor disturbances D_{100} and D_{300}. The beginning of the first rush hour leads to similar reactions in both cases. D_{300} gives two slightly higher peaks, and the metric reaches its maximum of $c_s \approx 0.0016$ at $t = 280$, while in D_{100}, the spike is $c_s \approx 0.0015$ at $t = 279$. In contrast, D_{500} only gives a peak of 0.007 at this point, only half as high. On the other hand, the higher number of cars in D_{500} leads to an earlier peak, which is at $t = 269$ in that case.

The second rush hour gives a more differentiated result. Here, the number of additional cars is reflected in the height of the peaks. D_{100} leads to a value of $c_s \approx 0.00075$ at $t = 779$, while D_{300} creates a speak of $c_s \approx 0.00125$ at $t = 779$. The biggest impact has D_{500}, where the metric reaches $c_s \approx 0.0016$ at $t = 783$.

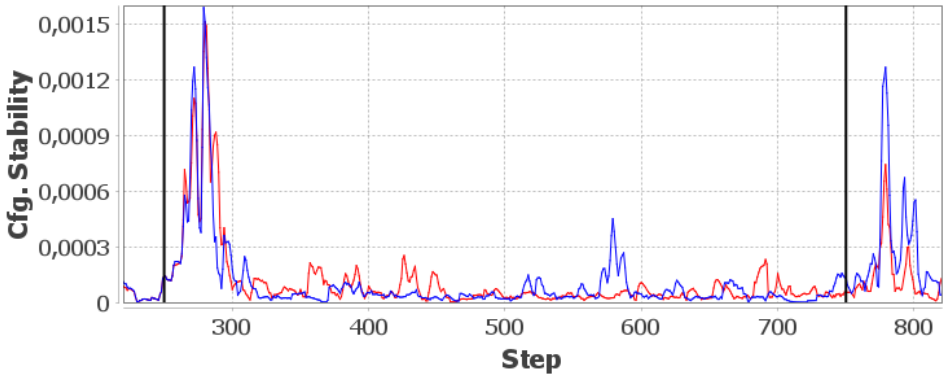


Figure 5.79: Detail view of the time series of the *Configuration Stability* for the Rush Hour scenario with parameters $M = L = \varepsilon = 10$ for D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

However, in all three cases, the general value level of $c_s \leq 0.0003$ is exceeded strongly at both rush hour events.

Configuration Variability

In Figure 5.80, we see the time series of the *Configuration Variability* for our two minor disturbances. The first rush hour with D_{100} leads to an increased baseline of $c_v \approx 1.5$ until $t = 370$. However, while the other two cases give a substantial spike at the beginning of the first event, D_{100} does not. Furthermore, D_{100} gives only a small peak with $c_v \approx 1.4$ until $t = 765$.

D_{300}, on the other hand, gives strong reactions in both rush hours. The first event leads to a peak with $c_v \approx 2.24$ at $t = 264$, and the peak in the second event reaches $c_v \approx 1.97$ at $t = 761$. This is comparable to the effects of D_{500}, where the *Configuration*

Variability reaches $c_v \approx 2.27$ at $t = 268$ and $c_v \approx 2.25$ at $t = 763$.

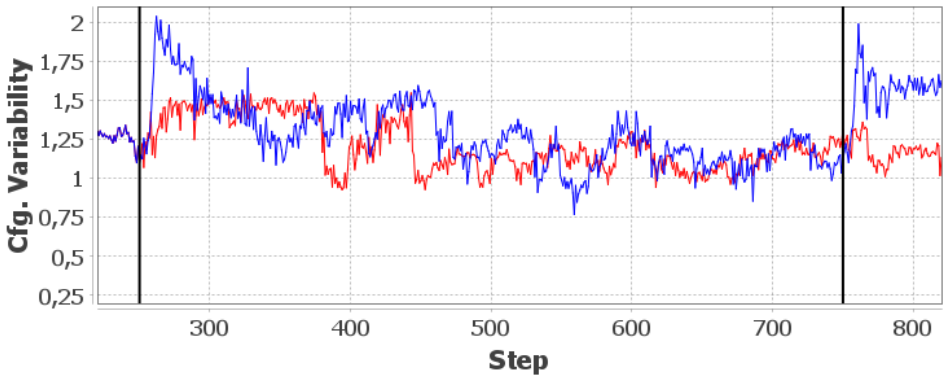


Figure 5.80: Detail view of the time series of the *Configuration Variability* for the Rush Hour scenario in the cases D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

Here, we see that the strength of the disturbance has a notable impact on the results of *Configuration Variability*.

Coherence Metrics

Figure 5.81 shows the result for the *Configuration Coherence*. In D_{100}, the metric shows no notable reaction to the first rush hour. It slowly decreases from the plateau at $coh_c \approx 0.06$, which started at $t = 247$, before the beginning of the rush hour, back to the baseline of $coh_c \approx 0.045$. The disturbance D_{300}, on the other hand, leads to a drop-down to $coh_c \approx 0.02$ at $t = 271$. At this point, D_{500} even reaches $coh_c \approx 0.015$.

The same differences appear during the second rush hour. D_{100} leads to a drop from $coh_c \approx 0.05$ to $coh_c \approx 0.033$ at $t = 757$ and reaches the previous level again at $t = 770$. In D_{300}, we see a drop from $coh_c \approx 0.05$ to $coh_c \approx 0.023$ at $t = 763$ and a longer stay at this level until $t = 883$. The substantial drop to a new baseline and the longer abidance there is comparable to the behaviour in D_{500}.

Figure 5.82 shows the result for the *Adaptation Coherence*. This metric shows the two rush hour events as substantial spikes in the time series and a volatile oscillation around the baseline of $coh_c \approx 0.3$ outside the rush hours. This applies to all three disturbance setups. However, the disturbances lead to different heights of the drops, where the strength of the disturbance corresponds to the height. In the first rush hour, D_{100} gives a drop to $coh_c \approx 0.16$, D_{300} gives $coh_c \approx 0.11$, and D_{500} leads to $coh_c \approx 0.09$. All the minima are at $t = 271 \pm 1$.

During the second rush hour, the behaviour is analogous: D_{100} gives a drop to $coh_c \approx 0.19$ at $t = 758$, D_{300} gives $coh_c \approx 0.14$ at $t = 760$, and D_{500} leads to $coh_c \approx 0.11$ at

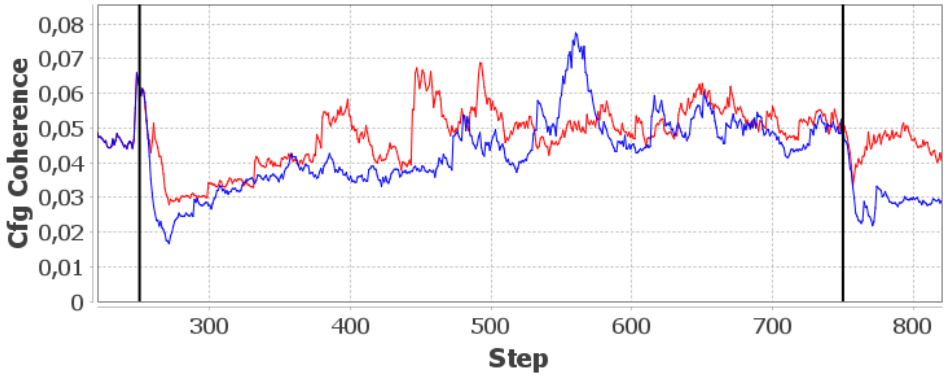


Figure 5.81: Detail view of the time series of the *Configuration Coherence* for the Rush Hour scenario in the cases D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

$t = 767$. Despite the difference in the time points, the metric returns to the baseline at $t = 776$ in all three cases.

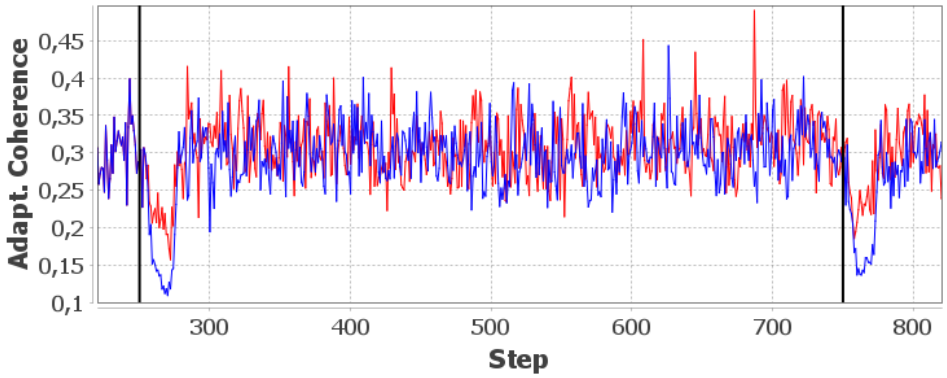


Figure 5.82: Detail view of the time series of the *Adaptation Coherence* for the Rush Hour scenario in the cases D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

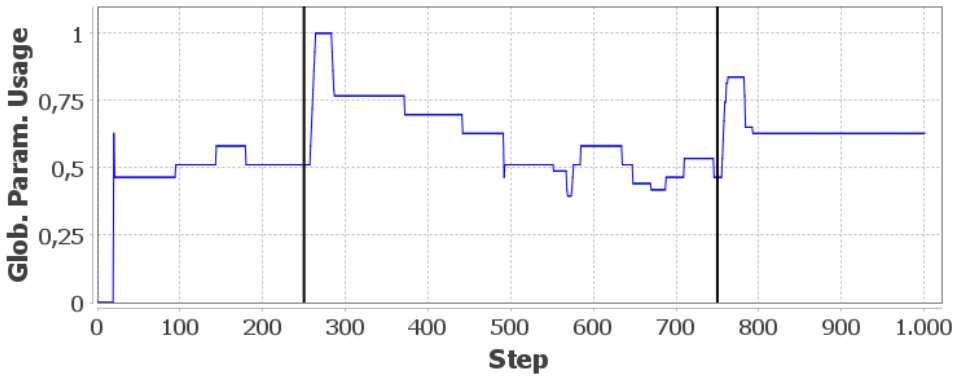
Parameter Usage Metrics

We will use $M = 20$ to analyse the differences between the three different disturbances. The main effect that the different disturbances have on the adaptation process is the range of values that the intersections use to steer the traffic flow. Every time step, the intersections are configured to let a specific number of cars go through for each lane. In D_{500}, the maximum of this number is 55 cars. For D_{300}, the maximum queue length is only 43 cars, and in D_{100}, this value is only 29 cars. The minimum that is used during

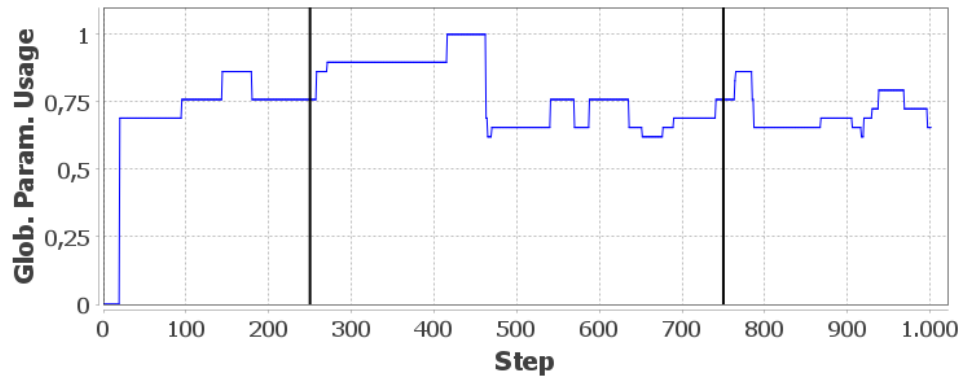
the simulations is 0. Since the interval length of the used parameters is a divisor in the metric, the time series show different baselines. Thus, the *Global Parameter Usage* for D_{100} starts at $U_g \approx 0.7$, and for D_{300} the initial value is $U_g \approx 0.49$.

Figure 5.83 shows the *Global Parameter Usage* for the two minor disturbances. We see that the first rush hour with D_{100} leads to a rise in the baseline but not to a spike. In D_{300}, we get an isolated peak, with a steep rise from $U_g \approx 0.5$ to $U_g = 1$. In the first rush hour, at least one intersection uses the maximum value for the queue length in this case. The same holds for D_{500}, where the initial baseline is at $U_g \approx 0.4$.

The second rush hour leads for D_{100} only to a small and short plateau of $U_g \approx 0.86$ from $t = 763$ to $t = 784$. In D_{300}, this event creates an increase from $U_g \approx 0.46$ to $U_g \approx 0.84$ with the same length. Here, the behaviour in D_{300} and D_{500} are qualitatively identical.



(a) The disturbance D_{300}



(b) The disturbance D_{100}

Figure 5.83: The time series of the *Global Parameter Usage* for the Rush Hour scenario with window size $M = 20$ in the two minor disturbances. The black lines denote the start of the two rush hours.

In Figure 5.84, we see the *Average Parameter Usage*. Since we are dealing with the averages of the value ranges here, the metrics result in the same initial values for all three disturbances and the same baseline with $U_a \approx 0.3$. The first rush hour event leads to comparable reactions in all three cases. D_{100} leads to a spike with $U_a \approx 0.515$ at $t = 274$, while the two bigger disturbances create a spike with $U_a \approx 0.525$ at $t = 282$.

The second rush hour shows a notable difference in the peak's height for D_{100}. For D_{100}, the *Average Parameter Usage* reaches $U_a \approx 0.55$ at $t = 775$; for D_{300}, it is $U_a \approx 0.65$ at $t = 776$, and D_{500} leads to $U_a \approx 0.66$ at $t = 777$.

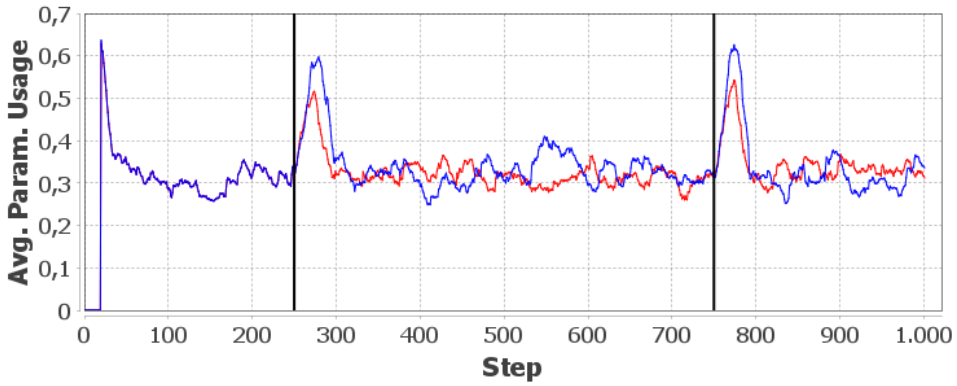


Figure 5.84: The time series of the *Average Parameter Usage* for the Rush Hour scenario with window size $M = 20$ in the cases D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

Configuration Divergence

Figure 5.85 presents the *Configuration Divergence* time series. This metric shows no notable reaction to the disturbance D_{100}. There is only a small peak of $c_d \approx 0.2$ at $t = 265$, which is smaller than the other spikes resulting from the background noise in the simulation. The second rush hour also gives no substantial change in the time series.

On the other hand, D_{300} results in a prominent peak of $c_d \approx 2.1$ at $t = 266$ and a peak of $c_d \approx 2.5$ at $t = 764$ for the second rush hour.

The reaction to the disturbance D_{500} is even stronger. The first peak reaches $c_d \approx 2.45$ at $t = 266$ and the second is at $c_d \approx 3.98$ at $t = 766$.

This shows that the *Configuration Divergence* is sensitive to the strength of the disturbance. Nevertheless, this metric is still able to identify D_{100} if we change the parameters to $M = L = 1$, as we can see in Figure 5.86. Here, both rush hours are visible with prominent spikes.

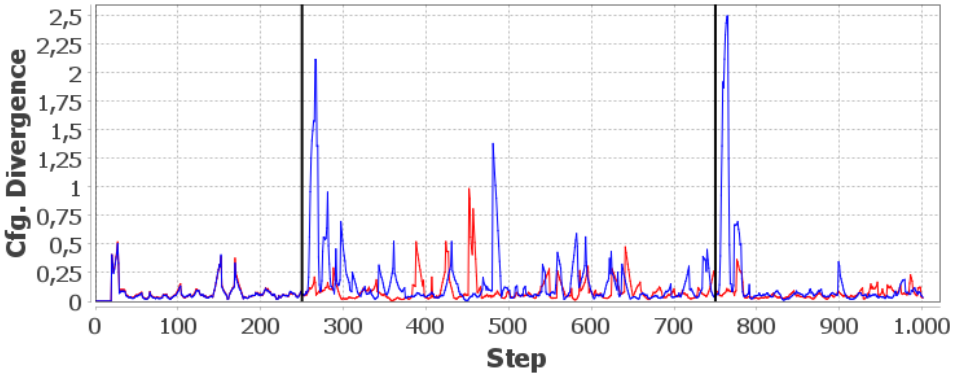


Figure 5.85: The time series of the *Configuration Divergence* for the Rush Hour scenario with $M = L = 10$ in the cases D_{300} (blue) and D_{100} (red). The black lines denote the start of the two rush hours.

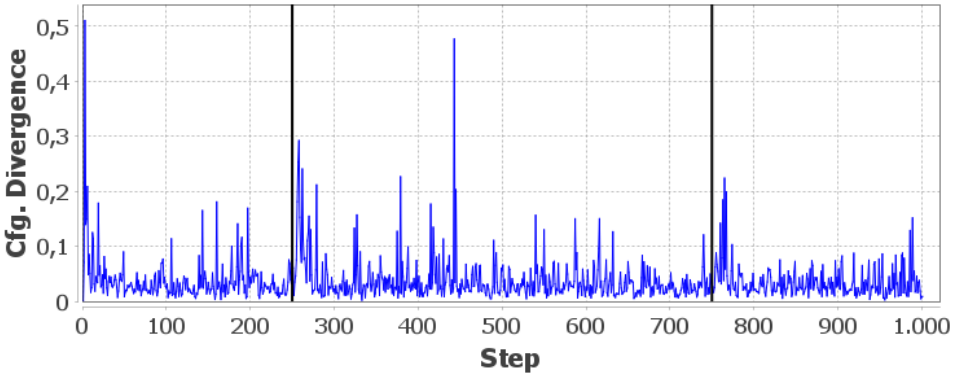


Figure 5.86: The time series of the *Configuration Divergence* for the Rush Hour scenario with $M = L = 1$ in the case D_{100}. The black lines denote the start of the two rush hours.

Summary

In this scenario, we see that the metrics show no common reaction pattern to changing strengths of the disturbance. While all metrics can identify both rush hour events in the medium disturbance D_{300}, the smallest disturbance D_{100} is visible only in the *Configuration Stability*, the *Average Parameter Usage* and both Coherence metrics, although the *Configuration Coherence* needs a specific configuration for this case.

5.2.7 Results for the Life-Like Road Network Scenario – OTC only

In this section, we see the results for the first two cases in the Life-Like Road Network scenario. We will compare the simulation with the artificial road blocks and the undisturbed simulation. The results for the simulation with activated DPSS are presented separately in Chapter 5.2.8 for a better overview.

The simulation runs with a resolution of 1 second per step. The times of the road blocks are marked with a grey background in the graphs.

The Aimsun simulator creates a log entry whenever a signal light changes its configuration. Since our metrics require data for every simulation step, the last seen values are repeated until an update occurs. This data extension ends with the last update entry. For the undisturbed case, the final adaptation takes place at 09:57:25 CET. In the simulation with the incidents, the last update happens 135 seconds later. Therefore, the metrics show 2 minutes and 15 seconds more data for the disturbed case. Since this part of the simulation is not of relevance to our analysis, we ignore this gap.

Entropy

The observed configurations take integer values in the range $[5, 108]$. Therefore, a maximum bin count of $M = 100$ seems an appropriate limit. Figure 5.87 shows the influence of the bin count on the *Entropy*. For all three values, the Entropy shows a reaction during the first blockade and after the start of the third one.

For example, with $M = 50$, the Entropy starts with $H \approx 3.32$, then reaches $H \approx 3.46$ at 07:11 CET and finally drops to $H \approx 3.17$ at 07:13 CET. During the third blockade, the metric gives another peak of $H \approx 3.36$ at 09:12 CET.

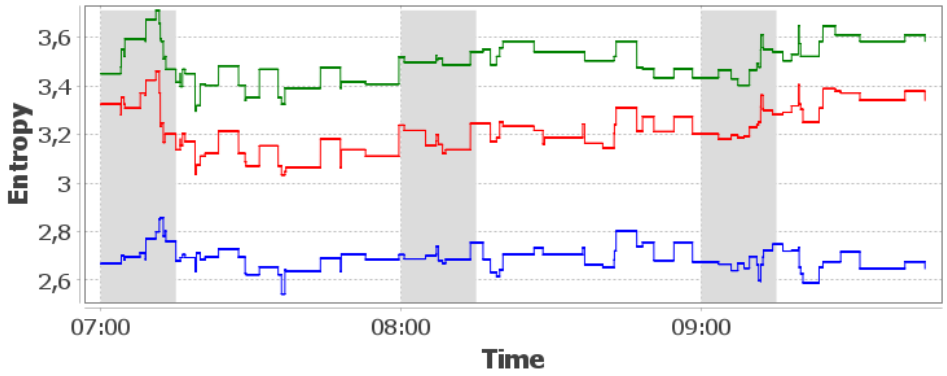


Figure 5.87: The time series of the *Entropy* for the Life-Like Road Network scenario in the disturbed case with bin counts $M = 20$ (blue), $M = 50$ (red), and $M = 100$ (green). The phases of the road blocks are marked grey.

Figure 5.88 compares the time series for the disturbed and the undisturbed simulation using $M = 25$. In the blockade simulation, the Entropy reacts with a delayed drop on the first incident and an immediate rise back to the baseline at $H \approx 2.17$. A similar reaction can be seen in the undisturbed simulation near t=07:25 CET. Since both reactions are comparable and since the other two blockades show no difference in the time series, we can state that the Entropy is not able to identify the events, not even when compared to the undisturbed case.

Furthermore, we note that the results for this choice for M give no better insight than those for the values in Figure 5.87.

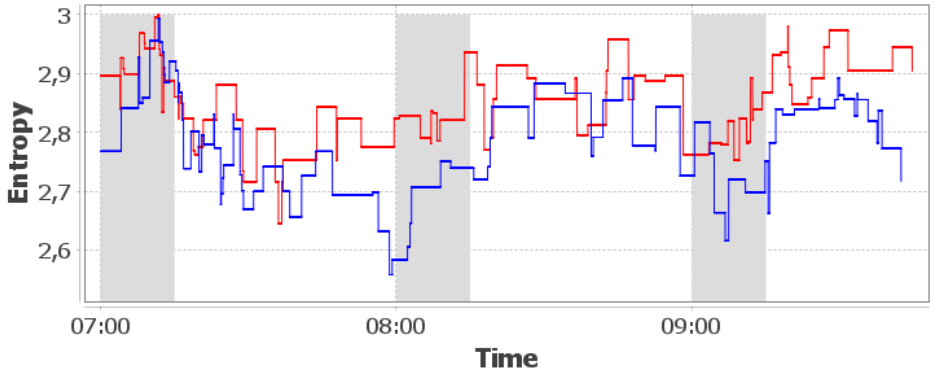


Figure 5.88: The time series of the *Entropy* for the Life-Like Road Network scenario with bin counts $M = 25$ for the undisturbed (blue) and the disturbed (red) simulation. The phases of the road blocks are marked grey.

Configuration Stability

For the *Configuration Stability*, first, we show the influence of the tuning parameters. Figures 5.89, 5.90 and 5.91 give an overview of how the parameters affect the results in the disturbed simulation.

In Figure 5.89, we see the influence of changing values for the window size M. For $M = 60s$ and $M = 120s$, the end of the first road block leads to a spike at 07:16 CET. With $M = 120s$, the time series reaches $c_s \approx 0.025$, and $= 60s$ gives a string spike of $c_s \approx 0.041$. $M = 180s$ generates no outstanding peak at this point. The third event gives a spike of $c_s \approx 0.03$ at 09:15 CET for $M = 60s$ and $c_s \approx 0.02$ at 09:14 CET for $M = 180s$. The medium window size $M = 120s$ creates a delayed spike of $c_s \approx 0.01$ at 09:18 CET. The second road block creates small peaks of $c_s \approx 0.005$ at 08:10 CET only for $M = 60s$ and $M = 120s$.

The comparison for the lag parameter L is shown in Figure 5.90. Higher values for the lag parameter L cause more spikes outside the road blocks. For $L = 120s$, the

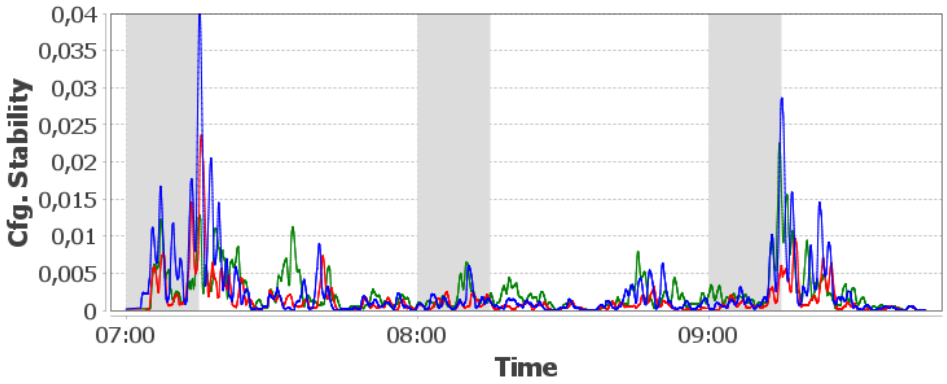


Figure 5.89: The influence of the tuning parameter M in the Life-Like Traffic scenario for the *Configuration Stability*. L and ε are fixed with $L = 120s$, $\varepsilon = 10$. Blue shows $M = 60s$, red is $M = 120s$ and green is $M = 180s$. The phases of the road blocks are marked grey.

highest spikes are shortly after the road block. Here, the *Configuration Stability* reaches $c_s \approx 0.025$ at 07:16 CET and $c_s \approx 0.01$ at 09:18 CET. By increasing L by 60s, we receive comparable spikes of $c_s \approx 0.018$ at 07:17 CET and $c_s \approx 0.011$ at 09:16 CET, and additional peaks, e.g. at 07:42 CET with $c_s \approx 0.014$. By changing L by another 60s to $L = 240s$, more peaks appear, e.g. the three strong spikes at 07:30, 07:36 and 07:43 CET.

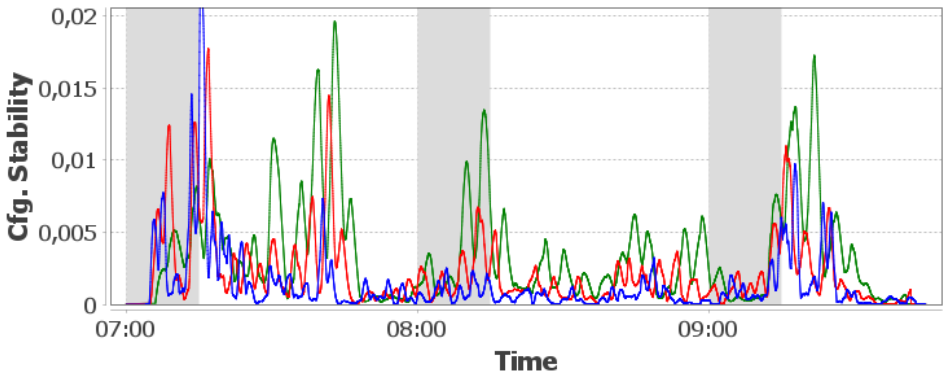


Figure 5.90: The influence of the tuning parameter L in the Life-Like Traffic scenario for the *Configuration Stability*. M and ε are fixed with $M = 120s$, $\varepsilon = 10$. Blue shows $L = 120s$, red is $L = 180s$ and green is $L = 240s$. The phases of the road blocks are marked grey.

Figure 5.91 compares the different values for the threshold ε. We see that this parameter only makes a difference in the height of the spikes. The peak at 07:16 CET reaches $c_s \approx 0.025$ for $\varepsilon = 10$, $c_s \approx 0.018$ for $\varepsilon = 20$ and $c_s \approx 0.01$ for $\varepsilon = 5$. However, with the

smallest value $\varepsilon = 5$, some spikes are notably higher than for the other values at certain points, e.g. at 08:12 CET with $c_s \approx 0.005$ or at 08:49 CET, again with $c_s \approx 0.005$. In fact, the range for suitable values for the threshold starts at $\varepsilon \approx 1.0$. Values bigger than $\varepsilon = 100$ still show the influence of the first and the third incident, but here, additional spikes start to grow higher than the interesting peaks.

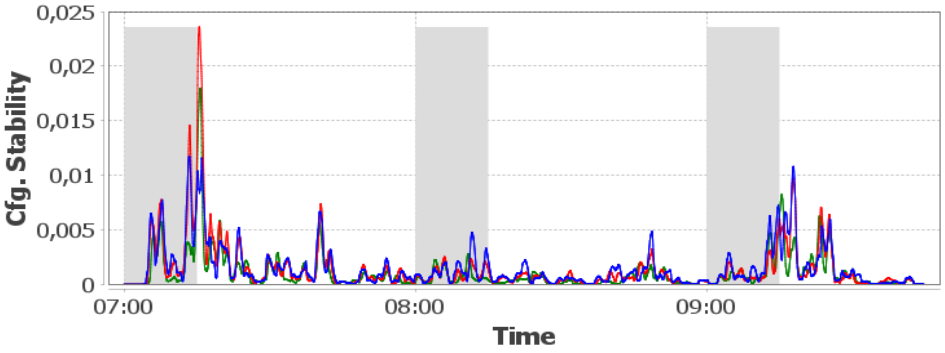


Figure 5.91: The influence of the tuning parameter ε in the Life-Like Traffic scenario for the *Configuration Stability*. M and L are fixed with $M = L = 120s$. Blue shows $\varepsilon = 5$, red is $\varepsilon = 10$ and green is $\varepsilon = 20$. The phases of the road blocks are marked grey.

An acceptable combination of parameters is $M = L = 120s$ and $\varepsilon = 10$. Applying the values to the two simulations gives us the times series in Figure 5.92. Compared with the undisturbed simulation, we see that the first incident creates a strong reaction in the *Configuration Stability*, leading to the spike of $c_s \approx 0.025$ at 07:16 CET. Also, the third disturbance is clearly visible in the rising values leading to the peak of $c_s \approx 0.01$ at 09:18 CET. In the undisturbed simulation, the adaptation behaviour leads to significantly smaller spikes. The maximum is reached at 07:22 CET with $c_s \approx 0.008$.

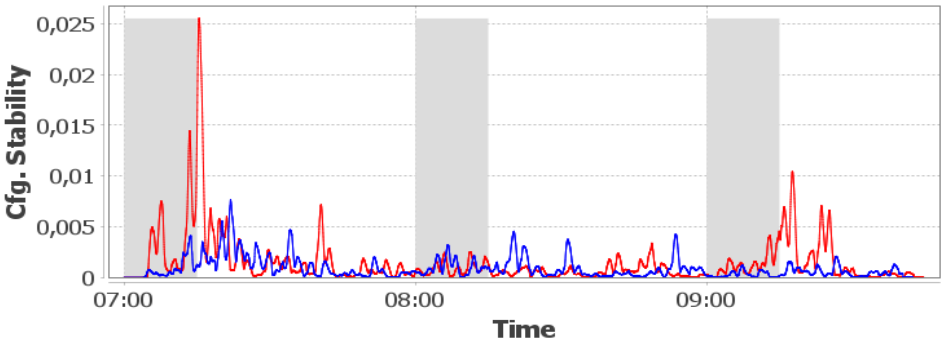


Figure 5.92: The time series of the *Configuration Stability* for the Life-Like Road Network scenario with parameters $M = L = 120s$ and $\varepsilon = 10$ for the undisturbed (blue) and the disturbed (red) simulation. The phases of the road blocks are marked grey.

Configuration Variability

In Figure 5.93, we see the *Configuration Variability* for the two simulations. Most values are not repeated from one evaluation step to the next. This leads to a very noisy graph. However, a general baseline can be identified. The main differences between the disturbed and the undisturbed case are the 30 minutes after the first and the third incident. Here, the values for the disturbed result stay in the range $c_v \in [6.6, 8]$ while the undisturbed time series remains at a baseline of $c_v \approx 7.25$ and above. Both time series show sudden, strong changes in the baseline. For the undisturbed simulation, examples include the plateau from 07:41 to 07:47 CET with $c_v \approx 7.2$ and the sharp drop at 08:39 CET down to $c_v \approx 6.2$. The simulation with the road blocks gives a similar drop to $c_v \approx 7.2$ at 08:37 CET and an outstanding plateau from 08:43 to 08:58 CET with $c_v \approx 7.5$. Comparing the behaviour in both simulations shows that the *Configuration Variability* taken alone cannot isolate any of the three disturbances.

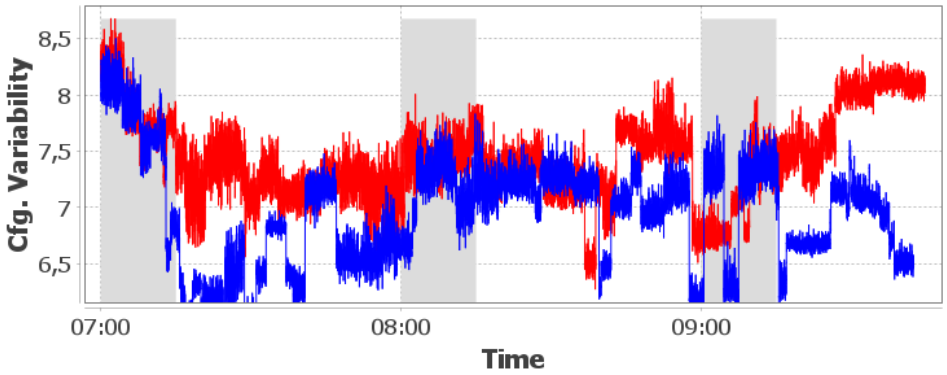


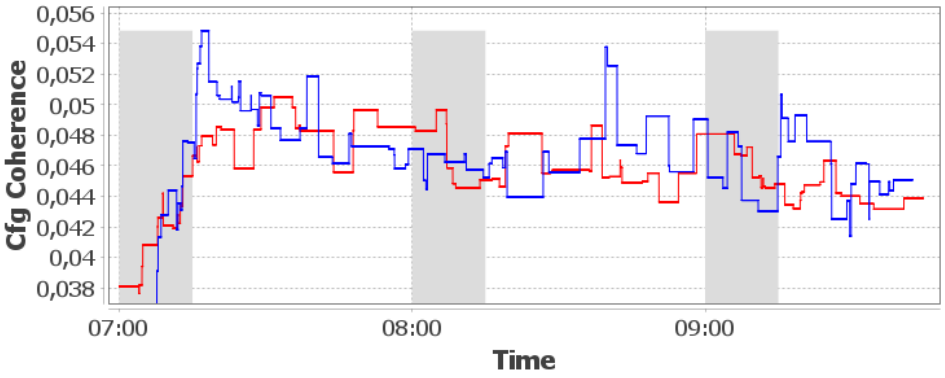
Figure 5.93: The time series of the *Configuration Variability* for the Life-Like Road Network scenario for the undisturbed (blue) and the disturbed (red) simulation. The phases of the road blocks are marked grey.

Coherence Metrics

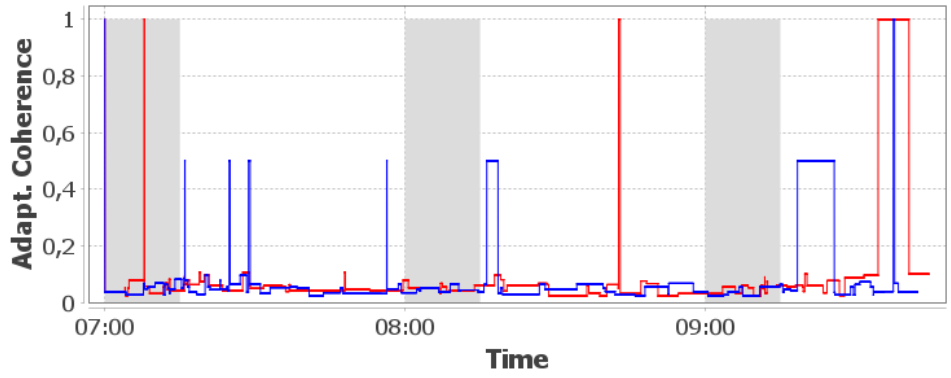
Figure 5.94 shows the times series of the *Coherence* metrics. For this scenario, we can state that these metrics do not give any indicators for the incidents. The *Configuration Coherence* has a comparable corridor between $coh_c \approx 0.044$ and $coh_c \approx 0.05$ in both simulations. Only the undisturbed simulation creates two spikes that exceed the upper bounds, one at 07:17 CET with $coh_c \approx 0.055$ and the second at 08:40 CET with $coh_c \approx 0.054$. Any impact of the incidents is not distinguishable from other events.

The *Adaptation Coherence* stays at a baseline of $coh_a \approx 0.05$ for both simulations, with only a few exceptions. The undisturbed case creates seven isolated spikes, with six

reaching $coh_a \approx 0.5$ and the last at 09:39 CET reaching $coh_a = 1$. The case with the incidents shows only three notable peaks, all with $coh_a = 1$. However, the second and the third spike do not relate to the road blocks.



(a) The Configuration Coherence

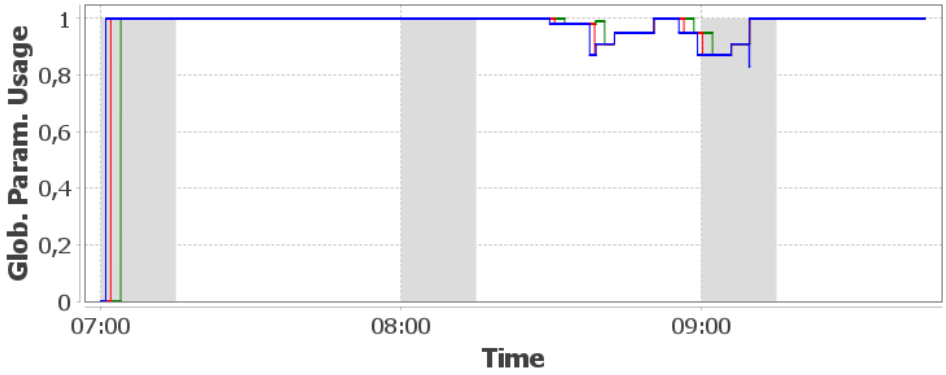


(b) The Adaptation Coherence

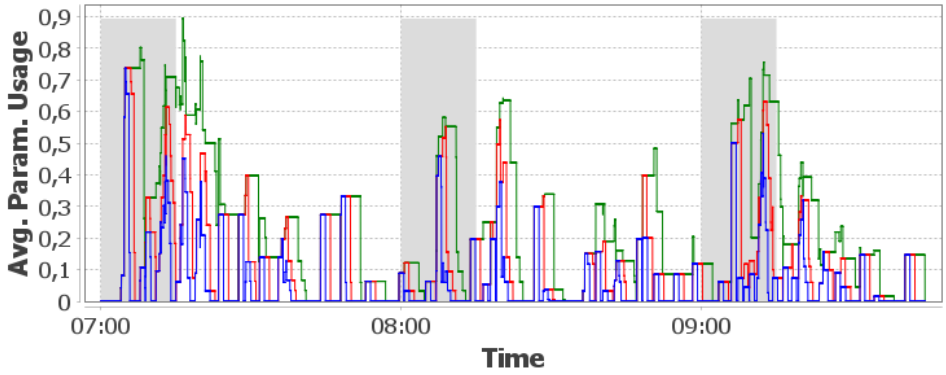
Figure 5.94: The times series of the *Coherence* metrics in the Life-Like Traffic scenario. Blue shows the undisturbed case, red is the disturbed case. The phases of the road blocks are marked grey.

Parameter Usage Metrics

Figure 5.95 shows the influence of the window parameter M on the *Parameter Usage* metrics. For both metrics, higher values for M lead to a delayed drop to lower metric levels. For example, the *Average Parameter Usage* reaches a plateau at $coh_a \approx 0.34$ at 07:48 CET which drops back to $coh_a \approx 0.34$ after M seconds. Rising metric values, on the other hand, appear immediately with all window sizes. Note that the small downward spike at 09:10:30 CET in the *Global Parameter Usage* vanishes for windows sizes $M > 65s$. It is then consumed by the increase at 09:10 CET.



(a) The Global Parameter Usage



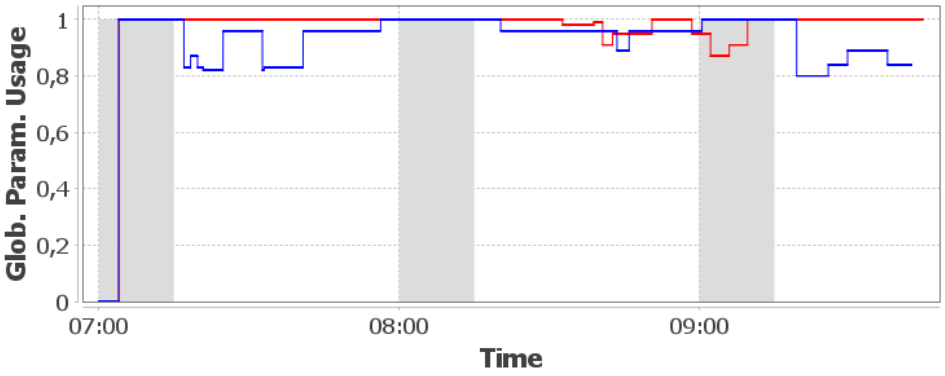
(b) The Average Parameter Usage

Figure 5.95: The times series of the *Parameter Usage* metrics in the Life-Like Traffic scenario for different window sizes M in the disturbed case. Blue is $M = 60s$, red is $M = 120s$, and green is $M = 240s$. The phases of the road blocks are marked grey.

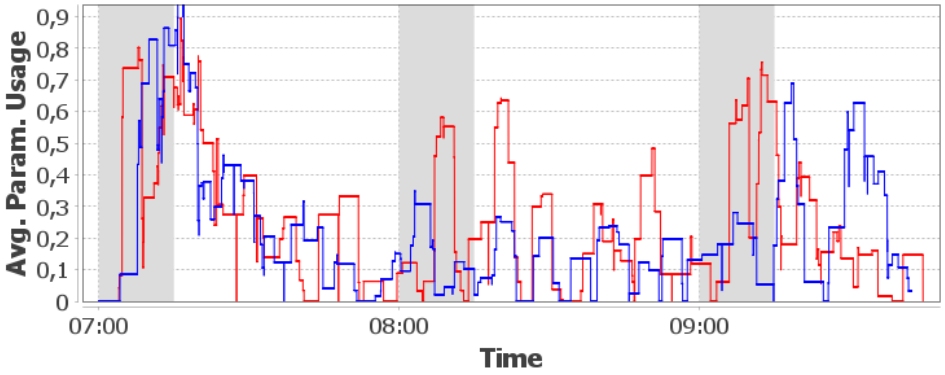
The comparison of the two scenarios can be seen in Figure 5.96. The *Average Parameter Usage* shows higher spikes between 8:00 and 9:15 CET in the disturbed simulation than in the undisturbed case. After that, it is the latter that gives the higher spikes. These spikes exceed $coh_a \approx 0.05$, while the general corridor is $coh_a \in [0, 0.3]$. During the first incident, both graphs show strong risings. In contrast, during and shortly after the second incident, the *Average Parameter Usage* shows stronger reactions in incident simulation than in the undisturbed case. Due to the reaction after 9:15 CET in the undisturbed case, we cannot clearly isolate the incidents in the graphs by comparing the two cases. The *Average Parameter Usage* is not able to identify the events in this scenario.

The *Global Parameter Usage*, on the other hand, shows a clear difference between the disturbed and the undisturbed case. While the undisturbed case shows a decrease from

$U_a = 1$ to $U_a = 0.8$ during several intervals, the disturbed case stays at $U_a = 1$ after all three incidents. When using the undisturbed case as a reference, the *Global Parameter Usage* is able to identify the end of the road blocks, although with a delay of several minutes.



(a) The Global Parameter Usage



(b) The Average Parameter Usage

Figure 5.96: The times series of the *Parameter Usage* metrics in the Life-Like Traffic scenario the two simulations with $M = 240s$. Blue is the undisturbed case and red the case with the incidents. The phases of the road blocks are marked grey.

Configuration Divergence

Figures 5.97, 5.98 and 5.99 show the evaluation of the *Configuration Divergence* in both simulations with different window sizes M and lags L. Higher values for M lead to delays in the reaction of the metric, wider bases for the spikes and generally higher values. In the undisturbed case, one example of the increasing delay can be seen in the two spikes at 07:59 and 08:02 CET for $M = L = 1s$. These spikes occur two minutes later when changing the parameters to $M = L = 120s$. Some time points create notable spikes for

all window sizes, e.g. at 08:52 and 08:56 CET (with $M = L = 120s$), which can be seen in the disturbed case. Nevertheless, there are no remarkable dissimilarities between the two simulations. The incidents do not lead to any identifiable differences.

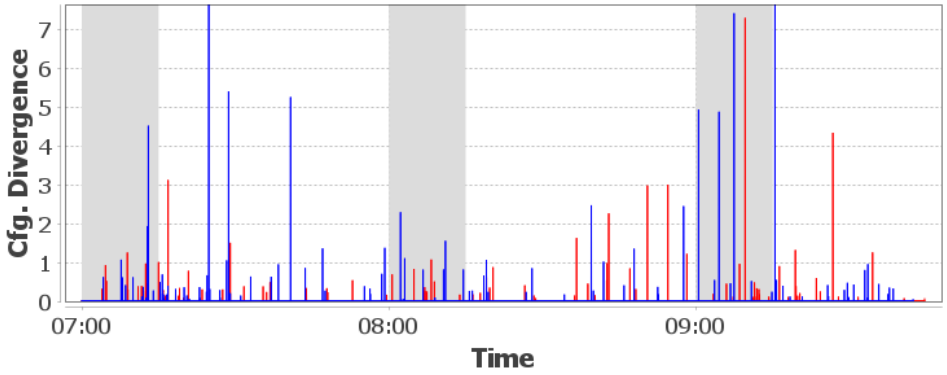


Figure 5.97: The *Configuration Divergence* in the Life-Like Traffic scenario for $M = L = 1s$. Blue is the undisturbed case and red the case with the incidents. The phases of the road blocks are marked grey.

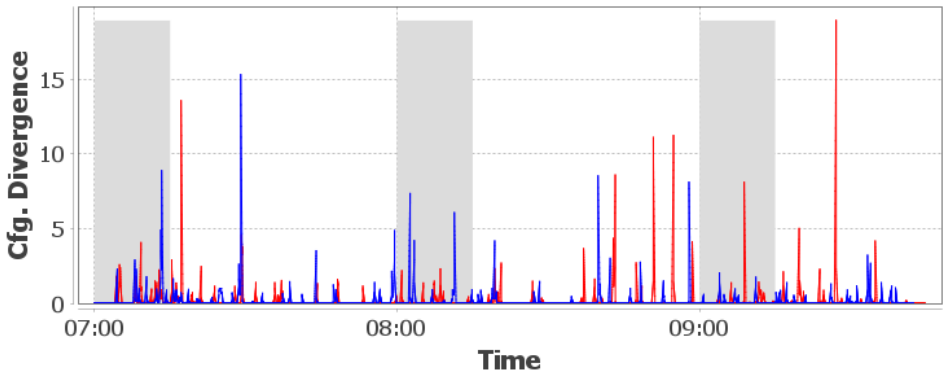


Figure 5.98: The *Configuration Divergence* in the Life-Like Traffic scenario for $M = L = 20s$. Blue is the undisturbed case and red the case with the incidents. The phases of the road blocks are marked grey.

Self-Organisation Divergence

The two cases do not implement any communication between the subsystems, and there is no mechanism that one subsystem gains knowledge about the state of another one. Therefore, the *Self-Organisation Divergence* is not applicable in this scenario.

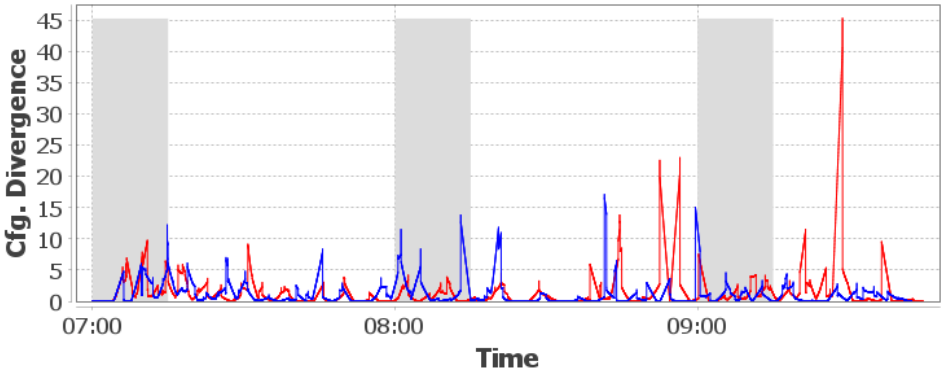


Figure 5.99: The *Configuration Divergence* in the Life-Like Traffic scenario for $M = L = 120s$. Blue is the undisturbed case and red the case with the incidents. The phases of the road blocks are marked grey.

Summary

In this scenario, only the *Configuration Stability* and the *Global Parameter Usage* are able to identify the first and the third incident. The *Configuration Stability* creates isolated peaks and can, therefore, detect the events directly without a comparison to an undisturbed time series. The *Global Parameter Usage*, on the other hand, needs this comparison to give an indicator for the events. Furthermore, the *Global Parameter Usage* only shows the end of the road blocks but not the beginnings.

All other metrics are unable to detect the events because the influence of the events cannot be separated from other configuration changes. Finally, the second road block at 9:00 CET creates not enough change in the configurations such that any metric could detect it.

5.2.8 Results for the Life-Like Road Network Scenario – OTC with DPSS

In this section, we see the results for the DPSS-activated case in the Life-Like Road Network scenario. Again, the simulation runs with a resolution of 1 second per step and the road blocks are marked with a grey background in the graphs. The influence of the tuning parameters for the metrics can be seen in Chapter 5.2.8. Thus, we restrict ourselves to only giving the results with those values for the parameters that have proved to be suitable.

This simulation incorporates direct communication between the subsystems. Therefore, we are able to use the *Communication Divergence* metric in this case. In total, the system creates 629 messages during the simulation.

Entropy

Figure 5.100 shows the *Entropy*. In this simulation, this metric gives no indication of any of the three blockage events. The activated DPSS leads to more oscillation in the time series compared to the OTC-only simulation.

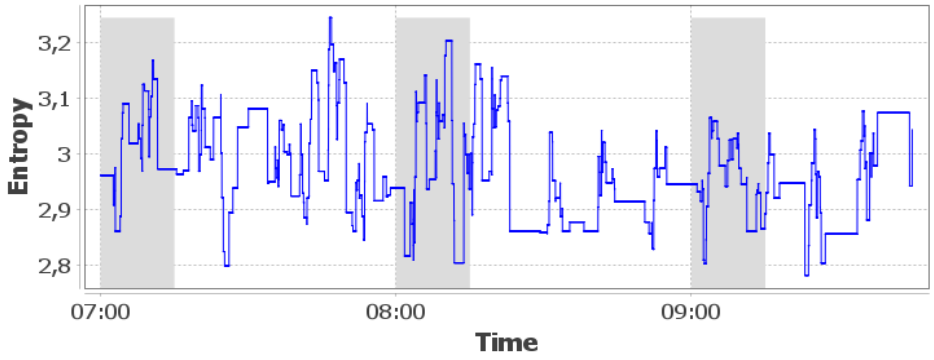


Figure 5.100: The time series of the *Entropy* for the Life-Like Road Network scenario with active DPSS. The bin count is $M = 50$. The phases of the road blocks are marked grey.

Configuration Stability

The *Configuration Stability* is depicted in Figure 5.101. Here, we can see a delayed impact of the beginning of the second and the third blockade. Both create isolated peaks in the time series, one at 08:10 CET with $c_v \approx 0.08$ and the second at 09:09 CET with $c_v \approx 0.07$. The first blockade gives no reaction.

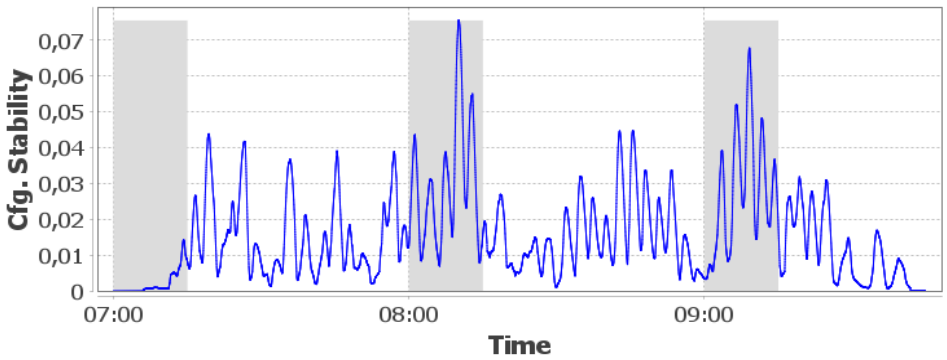


Figure 5.101: The time series of the *Configuration Stability* for the Life-Like Road Network scenario with enabled DPSS. $M = L = 180s$ and $\varepsilon = 10$. The phases of the road blocks are marked grey.

Configuration Variability

In Figure 5.102, we see the *Configuration Variability*. Here, the second event is identifiable due to an extraordinary spike from 08:09 to 08:11 CET with $c_v \approx 12.5$. The spikes during the other events are not distinguishable from those outside the event times.

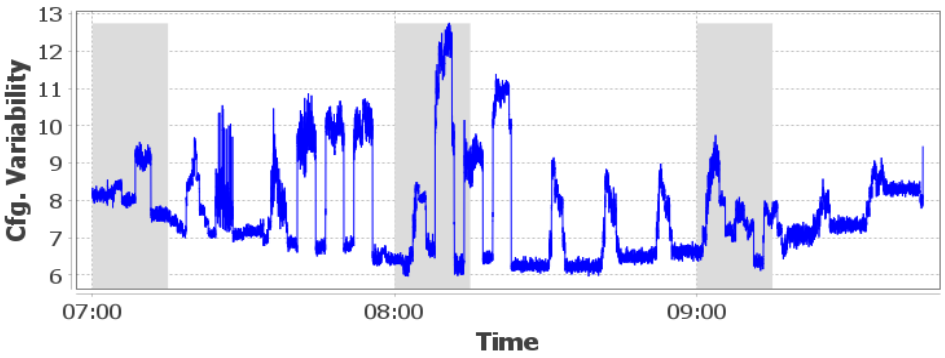


Figure 5.102: The time series of the *Configuration Variability* for the Life-Like Road Network scenario with activated DPSS. The phases of the road blocks are marked grey.

Coherence Metrics

Figures 5.103 and 5.104 show the times series of the *Coherence* metrics. As in the simulations without DPSS, both metrics do not give any indicators for the incidents in this case. The fact that the Adaptation Coherence shows several spikes with $coh_a = 1$ means that here all currently adapting intersections use the same value for their configuration. Although we see these spikes during the first and the second incident, we cannot infer a relationship. The spike at 07:27 CET and the two plateaus between 08:15 and 09:00 CET do not allow this.

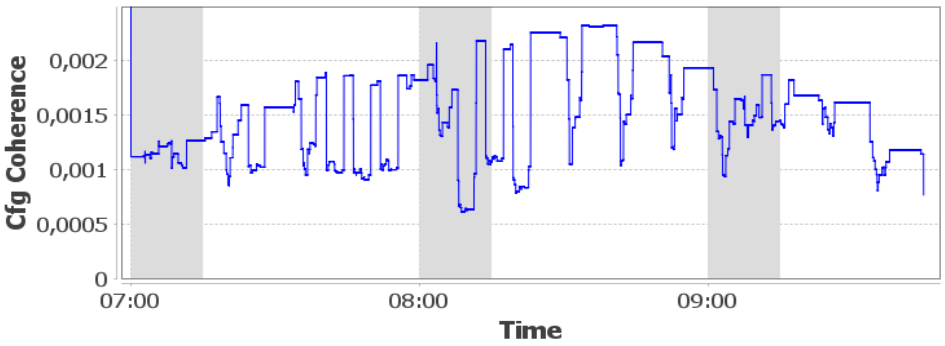


Figure 5.103: The times series of the *Configuration Coherence* in Life-Like Traffic scenario with activated DPSS. The phases of the road blocks are marked grey.

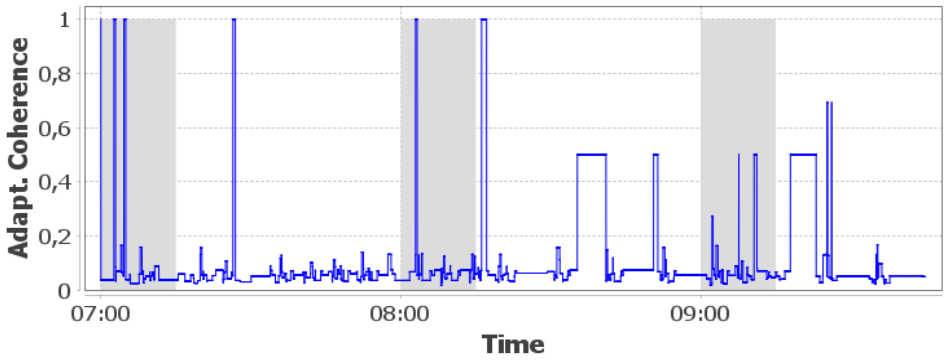


Figure 5.104: The times series of the *Adaptation Coherence* in Life-Like Traffic scenario with activated DPSS. The phases of the road blocks are marked grey.

Parameter Usage Metrics

The Figures 5.105 and 5.106 give the results for the two *Parameter Usage* metrics. Again, neither metric shows a distinguishable reaction to any of the three blockade events. All patterns inside the event frames can be found outside the frames.

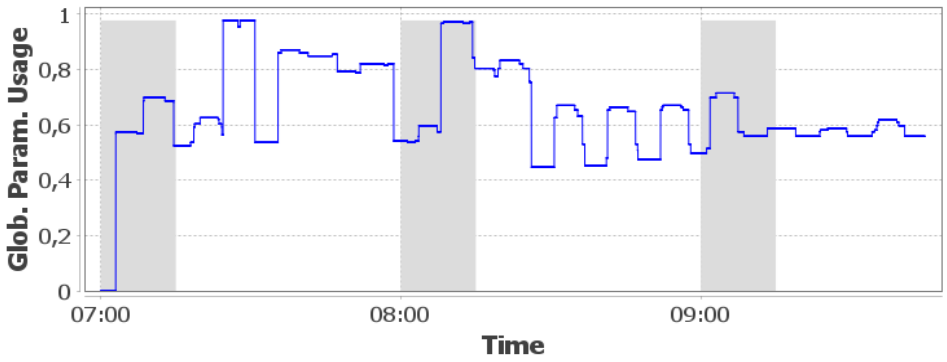


Figure 5.105: The times series of the *Global Parameter Usage* in the Life-Like Traffic scenario with activated DPSS. $L = 180s$. The phases of the road blocks are marked grey.

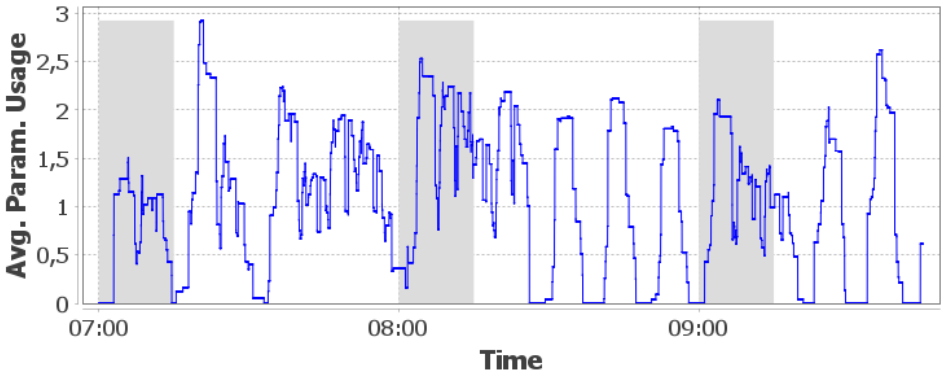


Figure 5.106: The times series of the *Average Parameter Usage* in the Life-Like Traffic scenario with activated DPSS. $L = 180s$. The phases of the road blocks are marked grey.

Configuration Divergence

Figure 5.107 shows the evaluation of the *Configuration Divergence*. Again, we see an influence of the second and the third event but none of the first one. Both detected events create isolated peaks. The second road block leads to two spikes of $c_d \approx 36.0$ at 08:06 and at 08:15 CET. The third road block gives a peak of $c_d \approx 36.0$ at 09:05 CET.

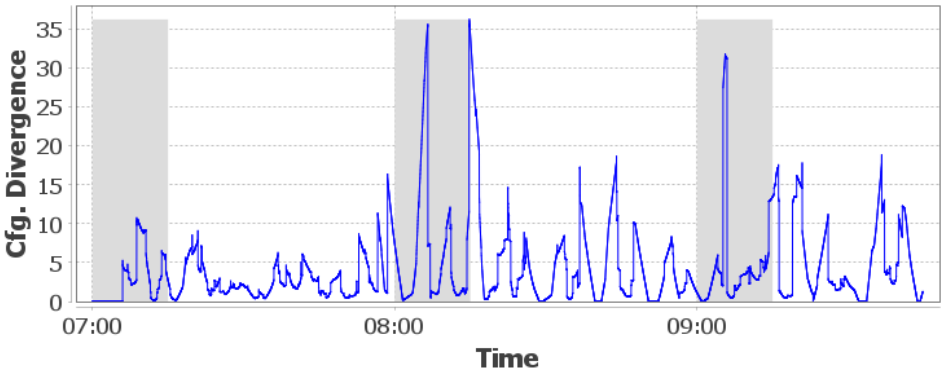


Figure 5.107: The time series of the *Configuration Divergence* in the Life-Like Traffic scenario with activated DPSS. $M = L = 180s$. The phases of the road blocks are marked grey.

Self-Organisation Divergence

Compared with the long period of three hours, the number of 629 messages in the system is small. The effect of this on the *Self-Organisation Divergence* can be seen in Figures 5.108, 5.109 and 5.110. This metric aims to measure the difference in the con-

tent of messages. With a frame size of 60 seconds, we only see that there are times with communication and times without. The difference in the messages themselves does not influence the graph. The longer the frame size is, the wider the spikes become, and eventually they start to merge. Only after that – when the time frame becomes long enough – the different message types can be regarded.

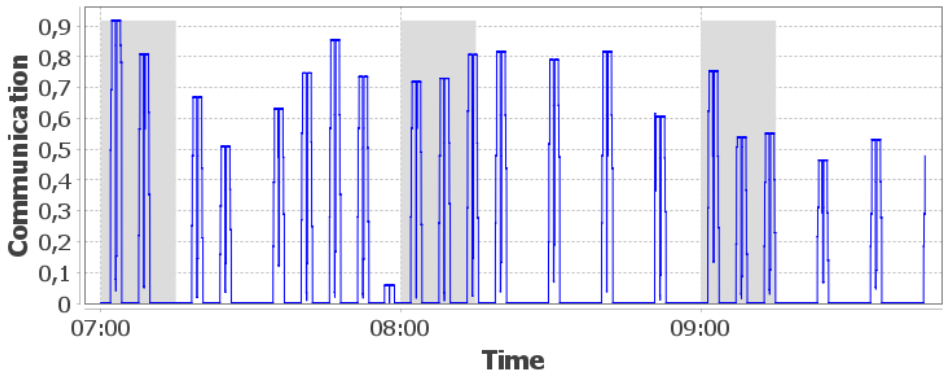


Figure 5.108: The times series of the *Self-Organisation Divergence* in the Life-Like Traffic scenario with activated DPSS for frame sizes $M = 60s$. The phases of the road blocks are marked grey.

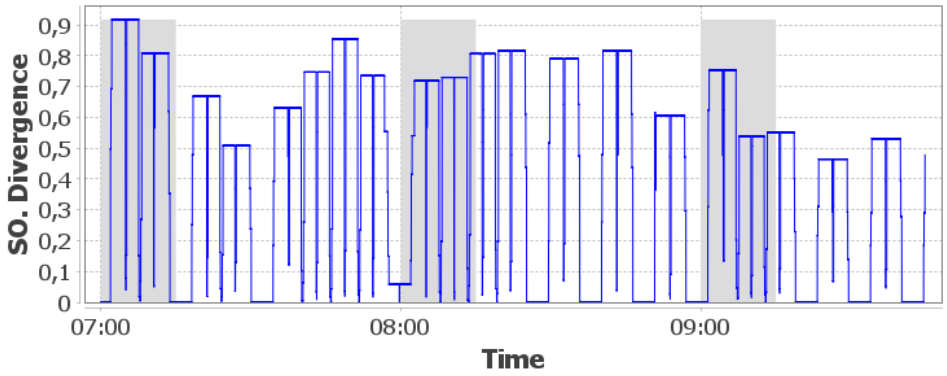


Figure 5.109: The times series of the *Self-Organisation Divergence* in the Life-Like Traffic scenario with activated DPSS for frame sizes $M = 180s$. The phases of the road blocks are marked grey.

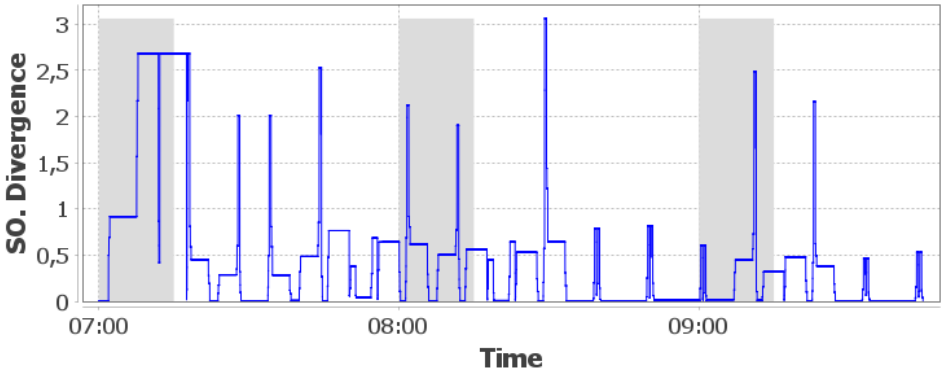


Figure 5.110: The times series of the *Self-Organisation Divergence* in the Life-Like Traffic scenario with activated DPSS for frame sizes $M = 600s$. The phases of the road blocks are marked grey.

Figure 5.111 shows the *Self-Organisation Divergence* for $M = 1800$ seconds. Here, we finally see the effect of the differing messages during the first 30 minutes. During this time, the intersections negotiate their configuration with the aim of optimising a system-wide traffic flow. On the other hand, the time series cannot differentiate between the general startup communication and the effects of the first road block. After this phase, the graph shows no further unusual patterns. The second and the third blockade event have no notable influence in this case.

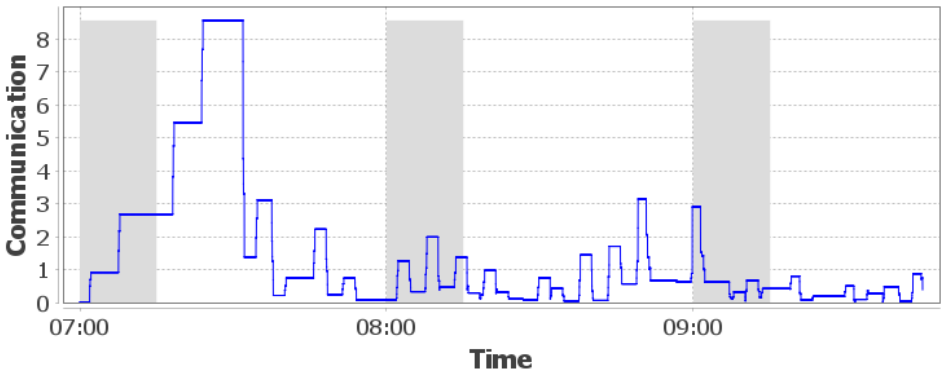


Figure 5.111: The time series of the *Self-Organisation Divergence* in the Life-Like Traffic scenario with activated DPSS for $M = 1800s$. The phases of the road blocks are marked grey.

Summary

As expected, the activation of DPSS leads to a different system behaviour. Only three metrics give indicators for the unusual events. The *Configuration Variability* shows the second road block, while the *Configuration Stability* and the *Configuration Divergence* additionally show the third event. The other metrics cannot separate the normal and the unusual situations. The *Self-Organisation Divergence* only shows the effects of the startup phase of the simulation but not the effects of the road blocks.

5.2.9 Results for the Maritime Traffic Scenario

In this section, we show the results of the Maritime Traffic Scenario with the data from the time frame with the accident and the frame before, when the Kiel Canal was in normal operation. The disturbance at 04:37 CET is marked with a black line in the figures. When interpreting the results of the metrics, we have to keep two important facts in mind: firstly, ships are not able to simply turn around in the canal (i.e. change their course immediately), and secondly, captains have to decide how to react on the canal closure: wait for the reopening or turn around and find another way to their destination. The duration of the closure is not clear in the first minutes after the accident. Therefore, we can assume that the accident appears with a notable delay in the metrics.

Note that the set of ships that are regarded in the two time frames are not the same. Ships that enter or leave the area during the respective time frame are considered still (or already) present with their last (if leaving) or first (if entering) course value. Therefore, the graphs for the second time frame do not necessarily start with the value that the first time frame ends with.

Entropy

The course over ground is given in the data set as a value in degrees in the interval $[0, 360)$. This leads to the assumption that a divisor of 360 would be a suitable bin count M. Figure 5.112 shows the results for $M = 90$, $M = 180$ and $M = 360$. We see that, in this case, these values give qualitatively identical graphs, only differing in the absolute value and in the height of the peaks. For example, the drop in between 23:00 und 23:30 CET is 5.2% for $M = 360$ and 7.6% for $M = 90$.

Figure 5.113 compares the time frame with the accident and the previous one with a bin count of $M = 90$. Our focus is on the time after 04:00 CET. Here, we see that the accident and the canal closure have only a small effect on the Entropy, where the values start to decrease from $H \approx 3.1$ to $H \approx 3.0$. Yet, compared with the normal operation, this decrease is barely noticeable. On the other hand, the strong drop from $H \approx 3.3$ to $H \approx 3.0$ near 23:00 CET is highly visible but does not correspond to any known incident.

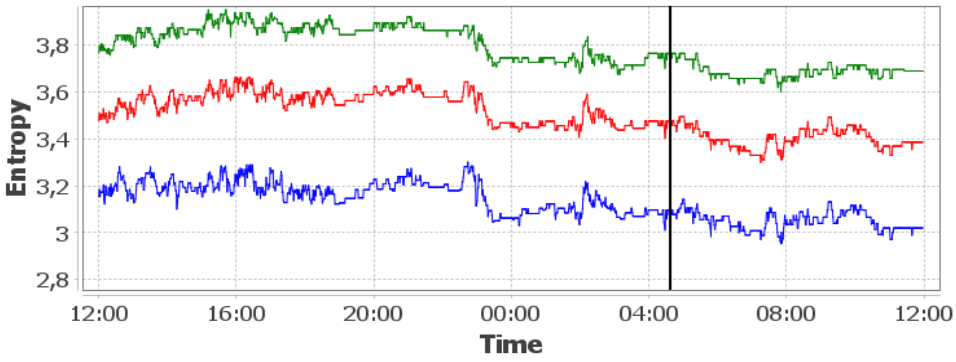


Figure 5.112: The time series of the *Entropy* for the Maritime Traffic scenario in the second time frame with bin counts $M = 90$ (blue), $M = 180$ (red), and $M = 360$ (green). The black line denotes the time of the accident.

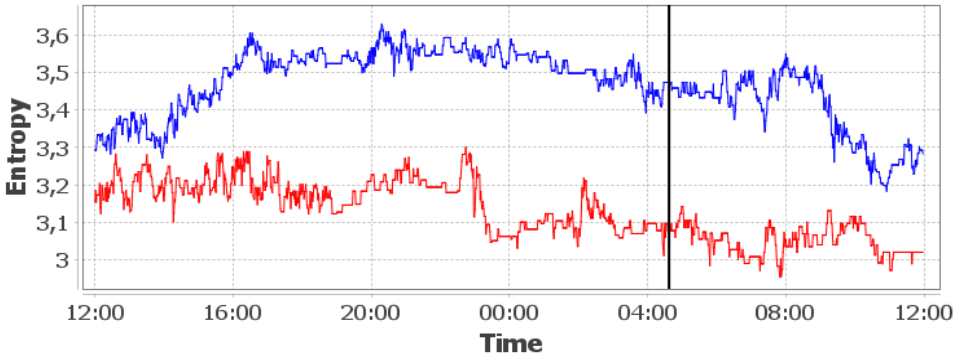


Figure 5.113: The time series of the *Entropy* for the Maritime Traffic scenario with bin count $M = 90$ for the normal operation time frame (blue) and the accident time frame (red). The black line denotes the time of the accident.

Configuration Stability

For the *Configuration Stability*, we will start with the influence of the tuning parameters M, L and ε.

Figures 5.114, 5.115 and 5.116 show the results for different values of the window size M. The impact of the canal closure can be seen better with smaller values. For $M = 10$ minutes, we have two peaks near 05:45 CET with $c_s \approx 0.00075$. $M = 30$ minutes also creates an isolated peak at this point with $c_s \approx 0.00013$, while $M = 20$ minutes leads to a peak that does not exceed other spikes before the event.

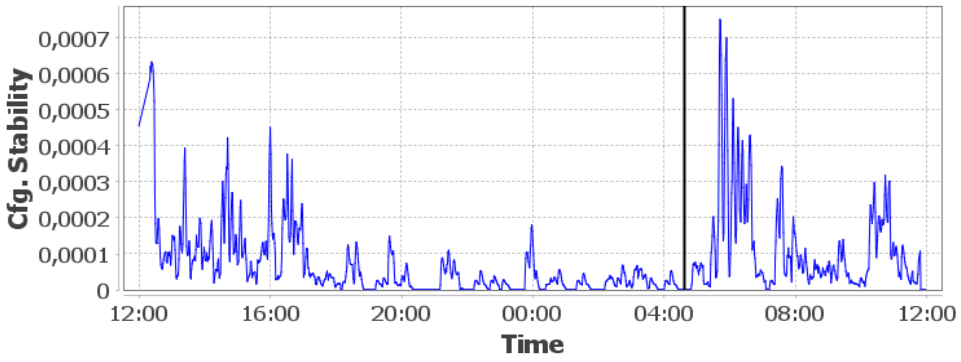


Figure 5.114: The *Configuration Stability* in the Maritime Traffic scenario with $L = 10$ minutes, $\varepsilon = 1$ and $M = 10$ minutes for the in the time frame with the accident. The black line denotes the time of the accident.

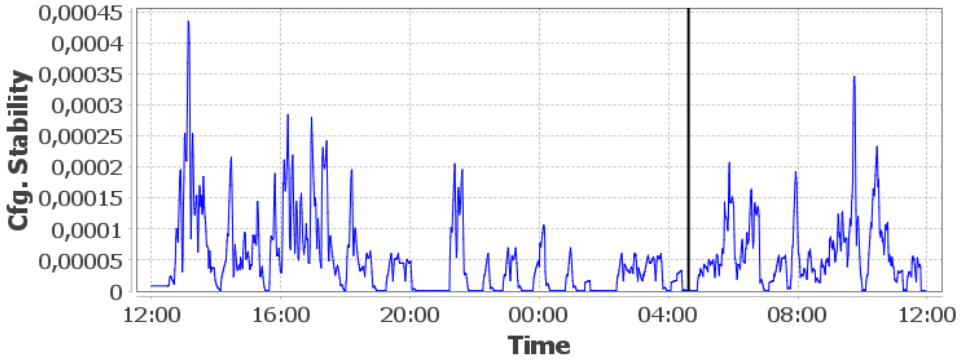


Figure 5.115: The *Configuration Stability* in the Maritime Traffic scenario with $L = 20$ minutes, $\varepsilon = 1$ and $M = 10$ minutes for the in the time frame with the accident. The black line denotes the time of the accident.

Figures 5.117, 5.118 and 5.119 show the results for different values of the lag parameter L. Again, the impact of the canal closure can be seen better with smaller values. The bases of the spikes become wider for larger L and start to merge with neighbouring spikes. Furthermore, for larger L, the height of the peaks that follow the accident decreases, and they align with the height of other peaks. For $L = 20$ minutes and $L = 30$ minutes, the time series start with a value above $c_s = 0.006$. For a better visibility, the Y-axis is cut off at the given values.

Finally, Figures 5.120, 5.121 and 5.122 show the results for different values of the threshold ε. Smaller values lead to a decreased height of the interesting peaks after the accident. For $\varepsilon = 2$, we see additional isolated spikes (e.g. at 00:00 CET with $c_v \approx 0.0002$). Therefore, $\varepsilon = 1$ is a suitable value for the analysis of this scenario.

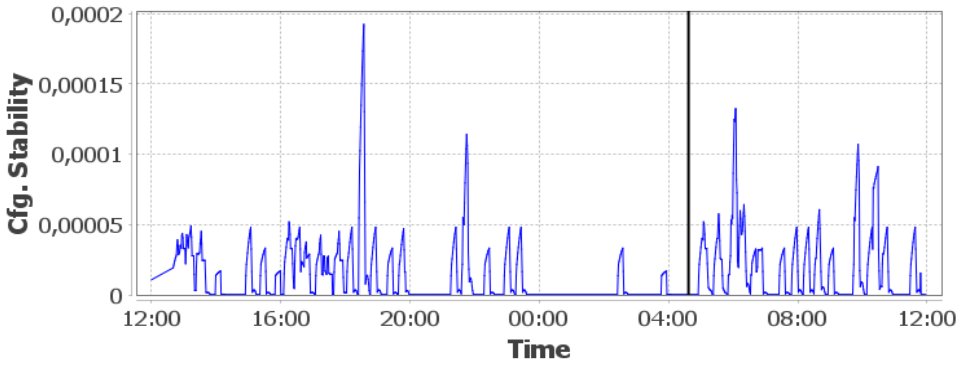


Figure 5.116: The *Configuration Stability* in the Maritime Traffic scenario with $L = 30$ minutes, $\varepsilon = 1$ and $M = 10$ minutes for the in the time frame with the accident. The black line denotes the time of the accident.

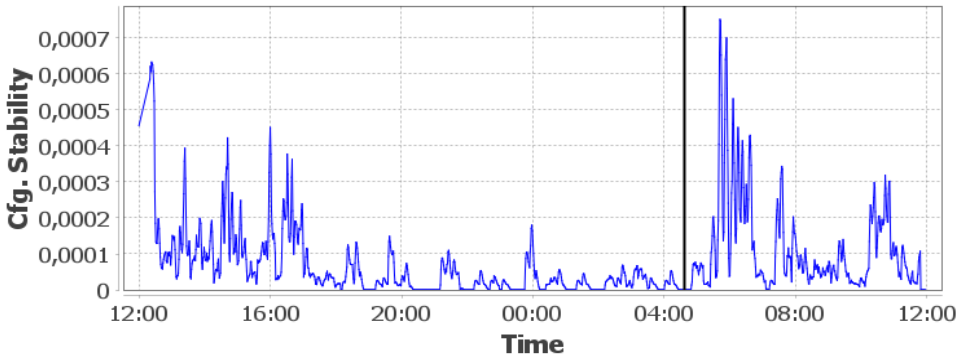


Figure 5.117: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = 10$ minutes, $\varepsilon = 1$ and $L = 10$ minutes. The black line denotes the time of the accident.

By choosing $M = L = 10$ minutes and $\varepsilon = 1$, we receive Figure 5.123. Here, we see the comparison of the two time frames. We can identify the canal blockade directly by the prominent peaks that start at $\approx$ 05:30 CET. This delay of nearly one hour is expected, as stated above. In the previous time frame, such a notable increase in the Configuration Stability starts only around 07:30 CET with a peak of $c_s \approx 0.0005$.

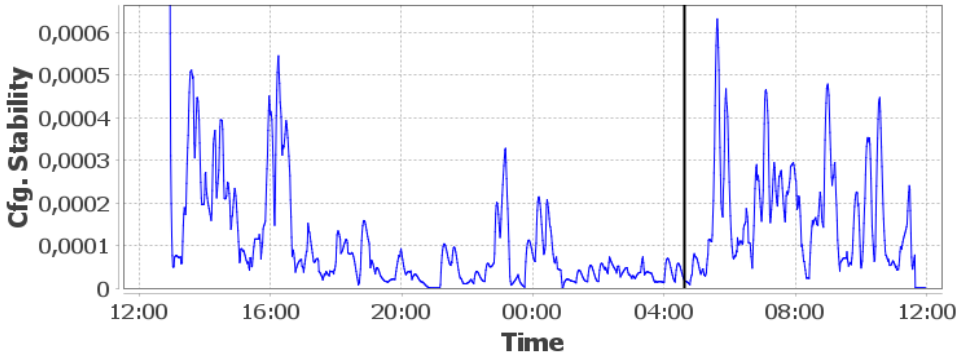


Figure 5.118: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = 10$ minutes, $\varepsilon = 1$ and $L = 30$ minutes. The black line denotes the time of the accident.

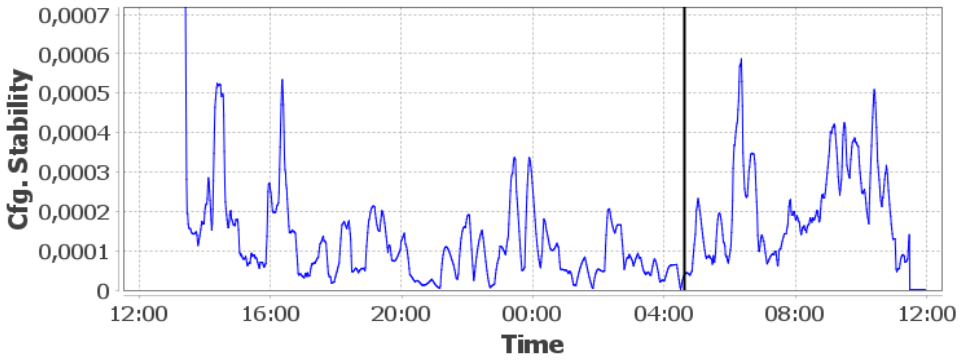


Figure 5.119: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = 10$ minutes, $\varepsilon = 1$ and $L = 30$ minutes. The black line denotes the time of the accident.

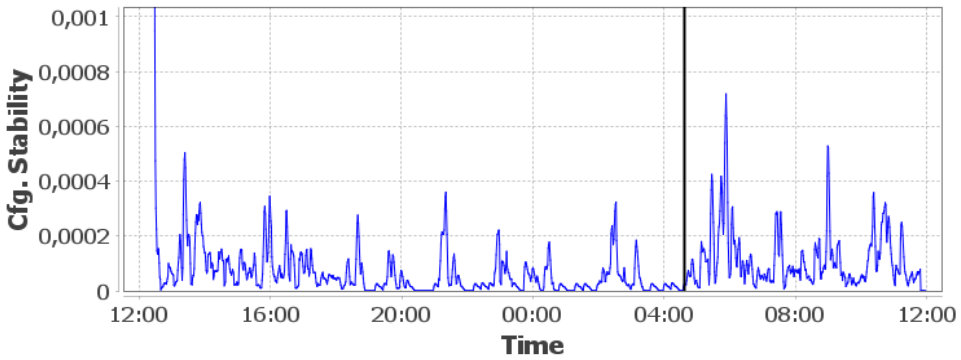


Figure 5.120: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = L = 10$ minutes and $\varepsilon = 0.5$. The black line denotes the time of the accident.

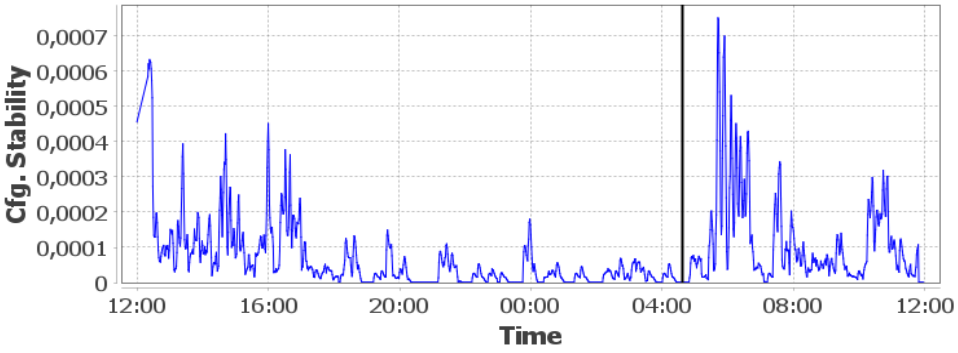


Figure 5.121: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = L = 10$ minutes and $\varepsilon = 1$. The black line denotes the time of the accident.

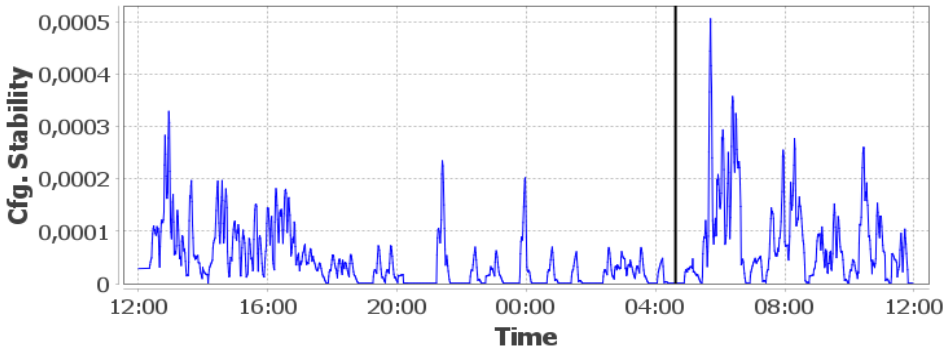


Figure 5.122: The *Configuration Stability* in the Maritime Traffic scenario in the time frame with the accident. $M = L = 10$ minutes and $\varepsilon = 2$. The black line denotes the time of the accident.

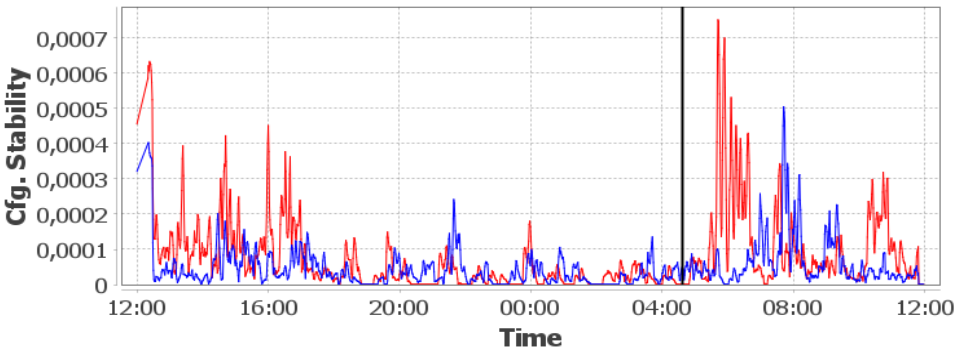


Figure 5.123: The time series of the *Configuration Stability* in the two time frames of the Maritime Traffic scenario. Blue is the first time frame with the normal operation, and red the frame with the accident. $M = L = 10$ minutes and $\varepsilon = 1$. The black line denotes the time of the accident.

Configuration Variability

Figure 5.124 shows the *Configuration Variability* in the two time frames. Both graphs are very noisy. The accident shows by a rise from $c_v \approx 0.38$ before the event to $c_v \approx 0.41$ with increased volatility after it. However, such changes also occur at other times, e.g. near 15:00 CET in the time frame of the accident.

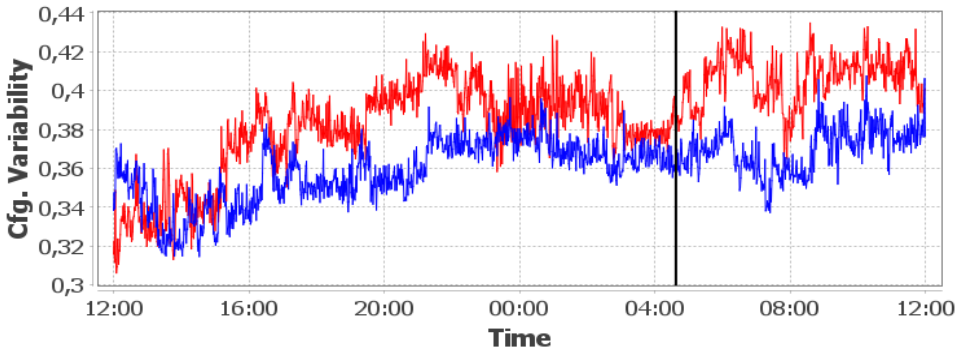


Figure 5.124: The time series of the *Configuration Variability* in the two time frames of the Maritime Traffic scenario. Blue is the first time frame with the normal operation, and red the frame with the accident. The black line denotes the time of the accident.

Coherence Metrics

The Figure 5.125 gives us a comparison of the *Configuration Coherence* in the two time frames. Here, the accident leads – with the expected delay of $\approx$ 1 hour – to a notable increase in coherence, a generally higher coherence level than in the time frame of the normal operation, and a noisier graph. During the reference period with the normal operation, increasing values for the *Configuration Coherence* occur only after 07:00 CET.

The *Adaptation Coherence* oscillates heavily between $coh_a = 0.5$ and $coh_a = 1$ without any clear patterns or isolated peaks. Therefore, Figure 5.126 shows only a section of the time series. What we see here happens everywhere. The reason behind this is the fact that in each time step only very few ships adapt their course, and in many steps, we only see one ship doing so. But, with only one data point, the statistical variance is zero, which leads to $coh_a = 1$. The fact that we use circular arithmetic to compute the mean and the variance of angles (see Chapter 4.1.4) leads to the lower bound of $U_a = 0.5$ when exactly two ships are involved, and their courses are directly opposite to each other.

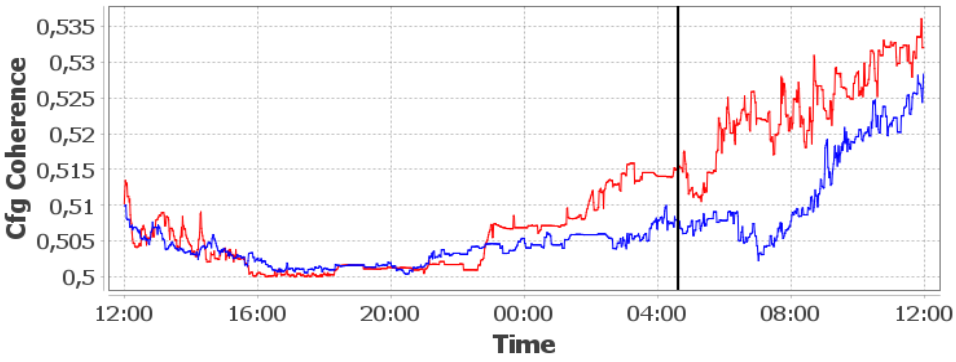


Figure 5.125: The time series of the *Configuration Coherence* in the two time frames of the Maritime Traffic scenario. Blue is the first time frame with the normal operation, and red the frame with the accident. The black line denotes the time of the accident.

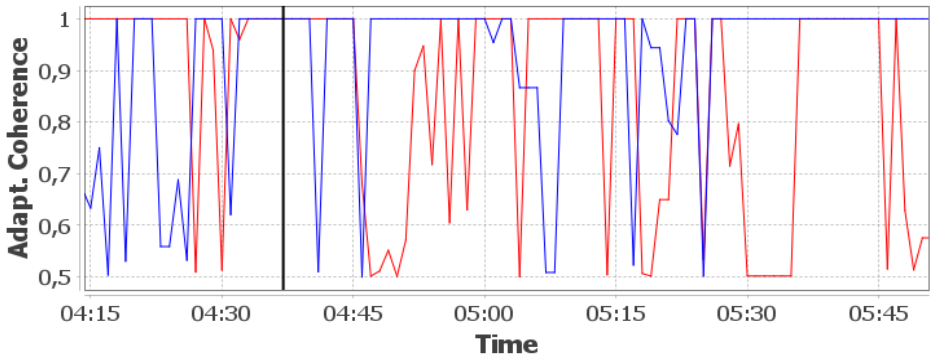


Figure 5.126: The time series of the *Adaptation Coherence* in the two time frames of the Maritime Traffic scenario. Blue is the first time frame with the normal operation, and red the frame with the accident. The black line denotes the time of the accident. Only a part is shown for a better overview.

Parameter Usage Metrics

Figures 5.127 and 5.128 show the influence of the window parameter M on the *Parameter Usage* metrics. For both metrics, higher values for M lead to a delayed drop to lower metric levels. Rising metric values, on the other hand, appear immediately with all window sizes. Furthermore, in the *Average Parameter Usage*, higher values for M lead to higher values for the metric.

The comparison of the two time frames with the *Average Parameter Usage* is shown in Figure 5.129. The impact of the accident is visible as the two highest peaks with values near $U_a \approx 0.04$. Since there are other isolated peaks with values $U_a \geq 0.03$, this metric needs to be compared with the reference time frame to give an indicator for the

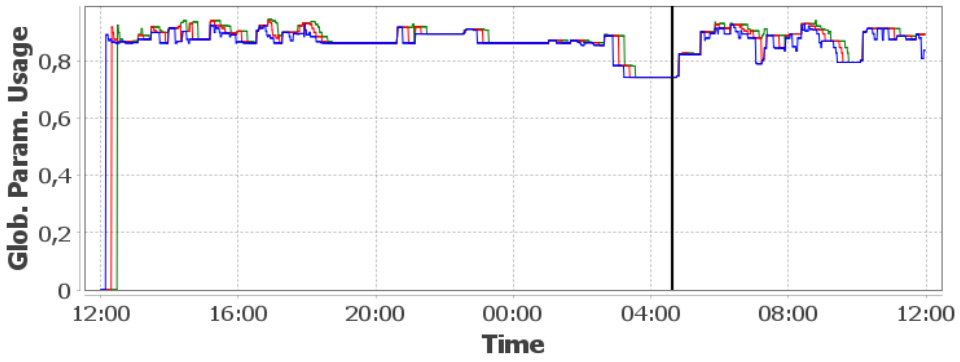


Figure 5.127: The times series of the *Global Parameter Usage* in Maritime Traffic scenario for different window sizes in the time frame with the accident. Blue is $M = 10$ minutes, red is $M = 20$ minutes, and green is $M = 30$ minutes. The black line denotes the time of the accident.

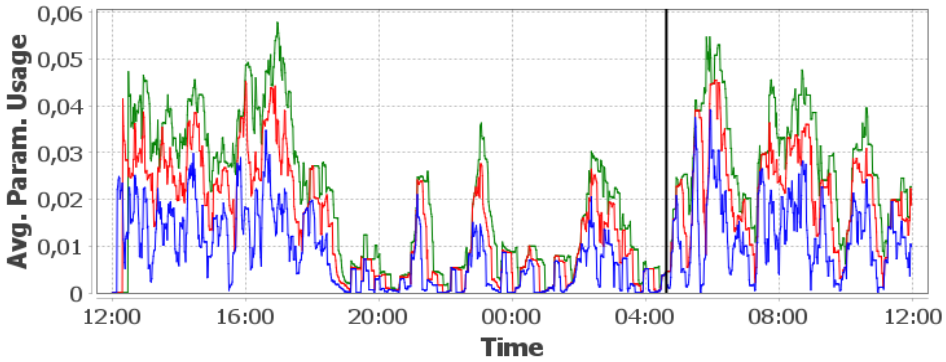


Figure 5.128: The times series of the *Average Parameter Usage* in Maritime Traffic scenario for different window sizes in the time frame with the accident. Blue is $M = 10$ minutes, red is $M = 20$ minutes, and green is $M = 30$ minutes. The black line denotes the time of the accident.

accident.

The behaviour of the *Global Parameter Usage* in the two time frames is given in Figure 5.130. Both time series start with a value of $U_g = 0$, which we cut off for a better overview. This metric shows an immediate reaction to the accident. Due to how this metric is calculated, course changes of a single ship can lead to a substantial change in the metric. In fact, this is the case here. While in the hour before the accident, all ships together only used 75% of the available 360 degrees for the course values, shortly after the accident, a few ships took new courses, which led to a greater range of used values.

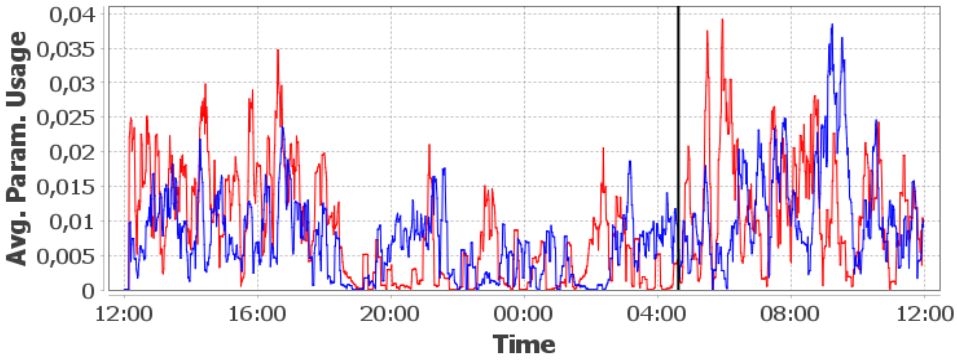


Figure 5.129: The times series of the *Average Parameter Usage* metrics in Maritime Traffic scenario in the two time frames. Blue is the first time frame with the normal operation, and red is the time frame with the accident. $M = 10$ minutes. The black line denotes the time of the accident.

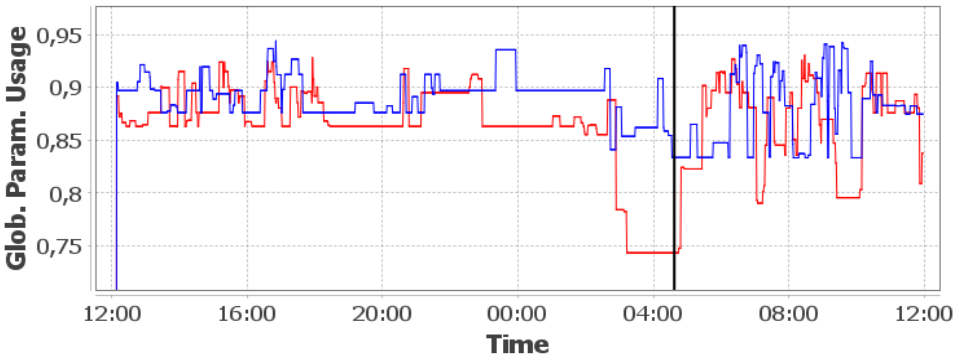


Figure 5.130: The times series of the *Global Parameter Usage* metrics in Maritime Traffic scenario in the two time frames. Blue is the first time frame with the normal operation, and red is the time frame with the accident. $M = 10$ minutes. The black line denotes the time of the accident.

Configuration Divergence

In Figures 5.131, 5.132 and 5.133, the influence of the window size M on the *Configuration Divergence* is shown. Again, the bigger window sizes lead to wider bases of the spikes and smaller values for the metric. For example, the peak at 23:00 CET has a value of $c_d \approx 0.017$ with $M = 10$ minutes, while for $M = 30$ minutes it reaches only $c_d \approx 0.0055$, but is still the highest values in this configuration. The increasing width of the bases can be seen in the spike at 18:30 CET. For $M = 10$ minutes, it has a width of 10 minutes which stretches to ≈ 60 minutes with $M = 30$ minutes.

In Figures 5.134 and 5.135, we see the influence of the lag parameter L ($L = 10$

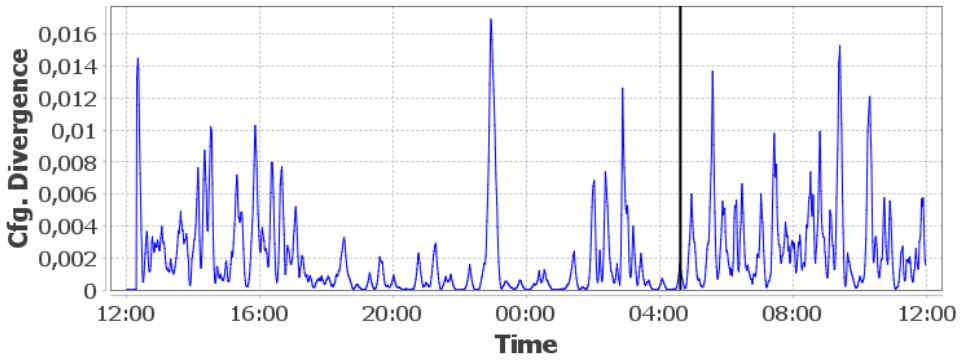


Figure 5.131: The *Configuration Divergence* in the Maritime Traffic scenario in the time frame with the accident. $L = 10$ minutes and $M = 10$ minutes. The black line denotes the time of the accident.

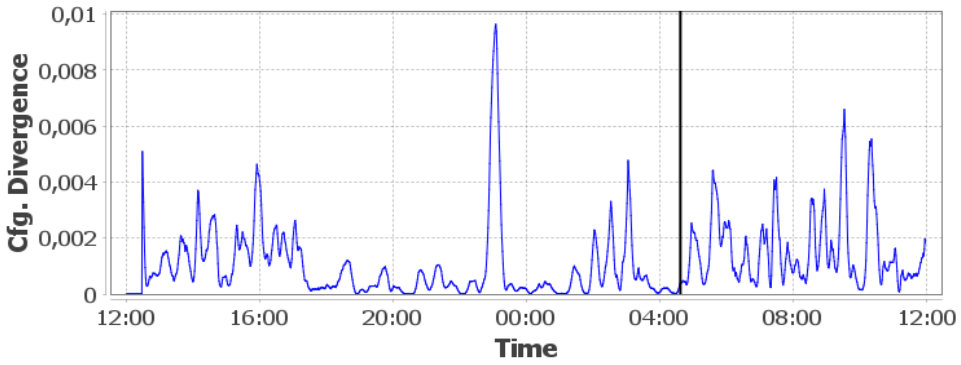


Figure 5.132: The *Configuration Divergence* in the Maritime Traffic scenario in the time frame with the accident. $L = 10$ minutes and $M = 20$ minutes. The black line denotes the time of the accident.

minutes is shown in Figure 5.131): higher values for L lead to bigger values for the metric. The peak at 23:00 CET with value $c_d \approx 0.0017$ for $L = 10$ minutes reaches $c_d \approx 0.0045$ for $L = 30$ minutes.

The *Configuration Divergence* is one of the metrics that reacts to the unknown event near 23:00 CET with an isolated peak. The accident, on the other hand, gives no identifiable reaction. Figure 5.136 shows the metric in the two time frames. During the time frame with the normal operation, we see increasing values in the morning hours after 07:00 CET. During this time, the values exceed $c_d = 0.003$. In the time frame with the accident, on the other hand, we see spikes after the accident, but also during the other times of the day. Thus, even in the direct comparison to the reference time frame, the accident is not visible.

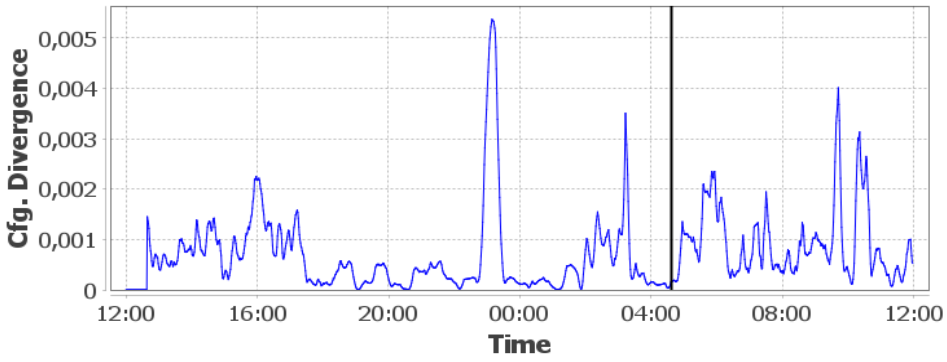


Figure 5.133: The *Configuration Divergence* in the Maritime Traffic scenario in the time frame with the accident. $L = 10$ minutes and $M = 30$ minutes. The black line denotes the time of the accident.

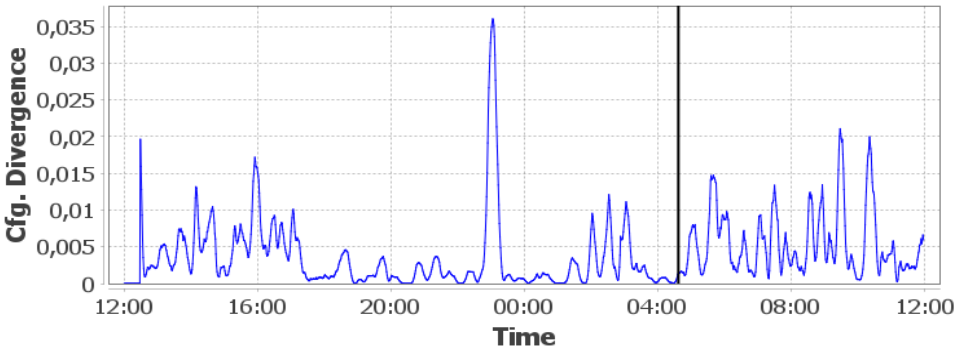


Figure 5.134: The *Configuration Divergence* in the Maritime Traffic scenario in the time frame with the accident with $M = 10$ minutes and $L = 20$ minutes. The black line denotes the time of the accident.

Self-Organisation Divergence

Communication data for the ships in this scenario is not available. Therefore, the *Self-Organisation Divergence* is not applicable in this scenario. Besides that, the bidirectional communication between ships is still usually a dialogue between humans, making the classification of message types and contents challenging.

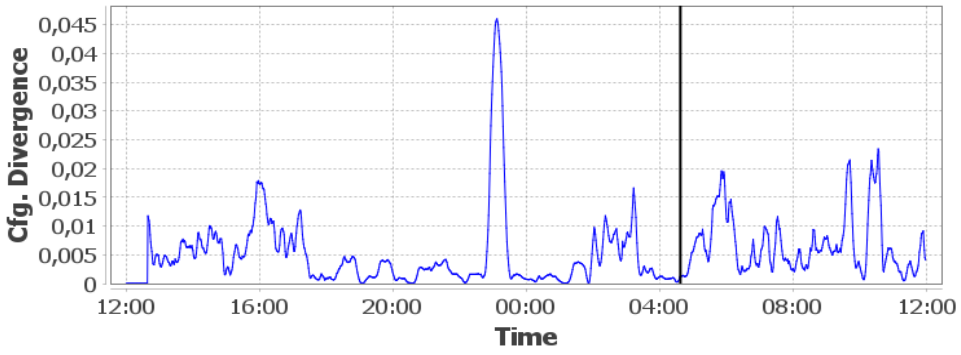


Figure 5.135: The *Configuration Divergence* in the Maritime Traffic scenario in the time frame with the accident with $M = 10$ minutes and $L = 30$ minutes. The black line denotes the time of the accident.

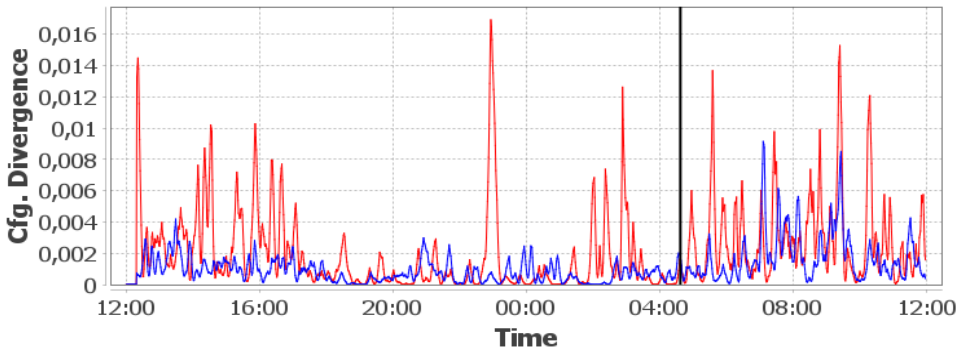


Figure 5.136: The times series of the *Configuration Divergence* metrics in Maritime Traffic scenario in the two time frames. Blue is the first time frame with the normal operation and red is the time frame with the accident. $M = L = 10$ minutes. The black line denotes the time of the accident.

Summary

The closure of the Kiel Canal can be identified with some delay by the *Configuration Stability*, both *Parameter Usage* metrics and the *Configuration Coherence*. Since the *Configuration Stability* creates unusually high spikes after the accident, it can be used directly. The other metrics require a comparison with the reference time frame to show the abnormal behaviour of the system. The other metrics were not able to give an indicator for the accident.

On the other hand, the *Entropy*, the *Configuration Stability* (with $M = 10$ and $L = 20$ minutes), the *Configuration Variability*, the *Configuration Coherence*, the *Average Parameter Usage*, and the *Configuration Divergence* give an indicator for an unknown event that occurred about 5:30 hours before the accident.

This scenario shows that our framework can be used to analyse AIS data to assess the collective behaviour of groups of maritime vessels in real-life situations.

5.2.10 Results for the Game of Life Scenario

This section gives the result of the Game of Life simulation. In this scenario, we have no disturbance. It runs for itself and simply follows the rules. Nevertheless, we observe two noteworthy events. Figure 5.137 shows the number of active cells over the course of time. The first interesting event occurs from $t = 154$ to $t = 169$, where the number of cells rapidly decreases from 420 to 243. The second event is the drop of active cells from 290 to 170, from $t = 212$ to $t = 225$. Now, let us see whether these events can be found in our metrics:

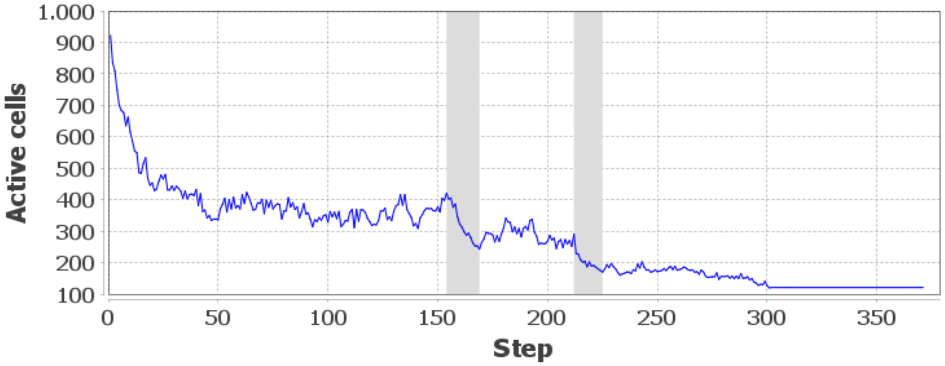


Figure 5.137: The number of active cells in the Game of Life simulation. The phases of interest are marked grey.

Entropy

Since the configuration in this system only allows for two values, a bin count of $M = 2$ is the only possible choice. Figure 5.138 shows the result. The simulation becomes stationary after $t = 302$, which leads to a constant *Entropy*. The decrease of the *Entropy* over time is also expected. The simulation starts with a chaotic behaviour and many state changes. Over time, patterns emerge, and the number of cells that change their state decreases.

At first glance, the graphs for the *Entropy* and the number of active cells (Figure 5.137) appear identical up to scaling. They are not! In Figure 5.139, we scaled both time series to the interval $[0, 1]$ and calculated the difference $D(t) = E(t) - |\{a : \text{a is active at t}\}|$. We see that the difference is relatively small, but the scaled *Entropy* is bigger than the scaled number of active cells most of the time.

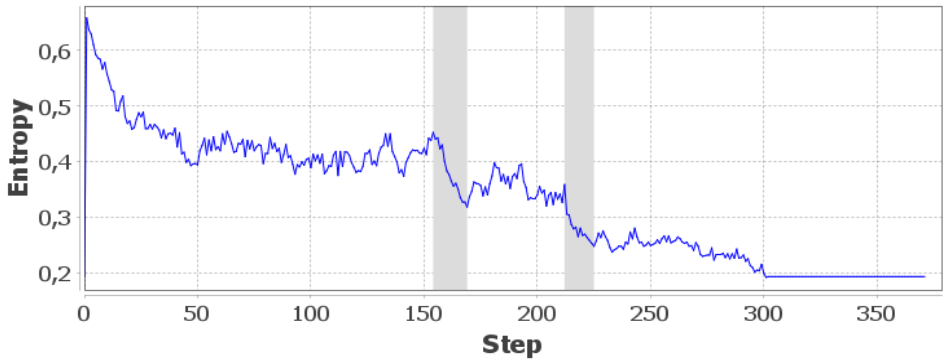


Figure 5.138: The time series of the *Entropy* in the Game of Life simulation with $M = 2$. The phases of interest are marked grey.

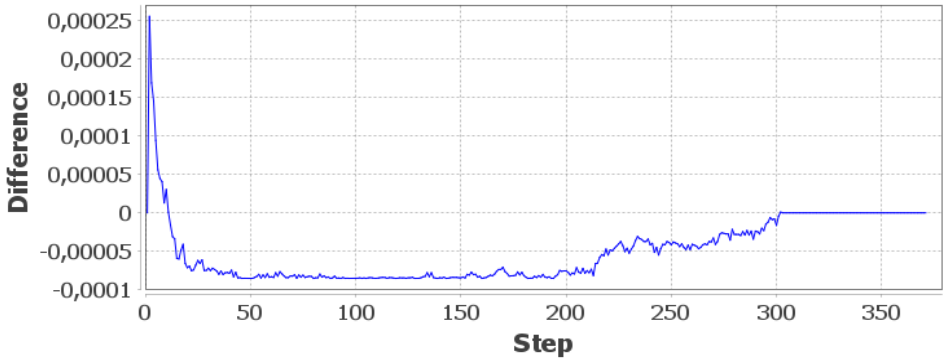


Figure 5.139: The time series of the difference between the scaled *Entropy* and the scaled number of active cells in the Game of Life scenario.

Configuration Stability

Since there is no disturbance in this simulation and no notable configuration changes emerge, we cannot search for suitable parameters that would allow an indication for such an event. Therefore, Figures 5.140, 5.141 and 5.142 give a few combinations. Again, as in the previous scenarios, the parameters influence the delay of peaks and the widths of their bases.

The first event at $t = 154$ is visible as a peak in all three graphs. Due to the metric's delay, the corresponding peak is at $t = 166$ with $c_s \approx 0.00004$ for $M = L = 5$. With $M = L = 20$, the main peak moves to $t = 194$ and reaches $c_s \approx 0.00007$ for $M = L = 5$. The second event is only visible in the two graphs with the higher values for M and L. The peak after $t = 200$ in the graph for $M = L = 5$ is not the result of the event. This peak comes too early. For $M = L = 20$, the second peak at $t = 166$ reaches $c_s \approx 0.00004$, while for $M = L = 10$, we get $c_s \approx 0.00001$ at $t = 231$.

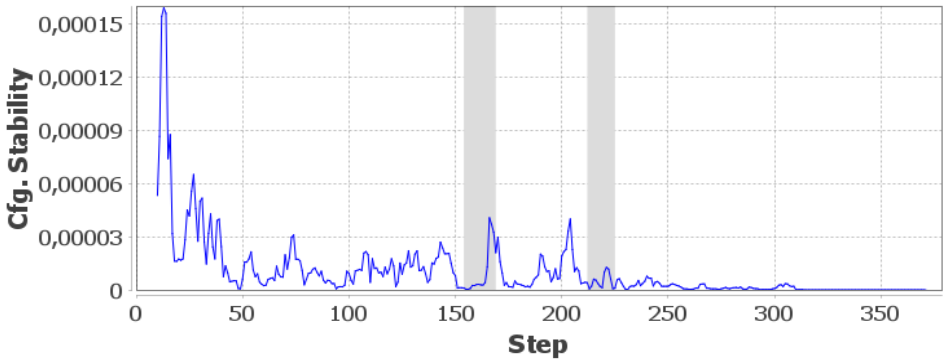


Figure 5.140: The *Configuration Stability* in the Game of Life simulation with $M = L = 5$ and $\varepsilon = 0.025$. The phases of interest are marked grey.

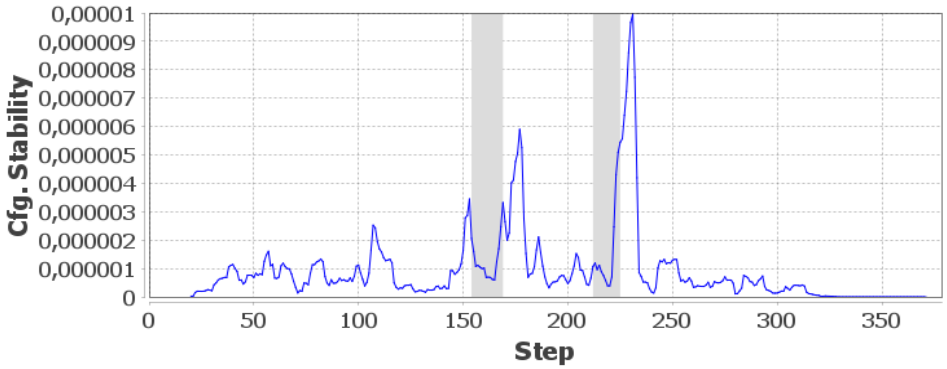


Figure 5.141: The *Configuration Stability* in the Game of Life simulation with $M = L = 10$ and $\varepsilon = 0.05$. The phases of interest are marked grey.

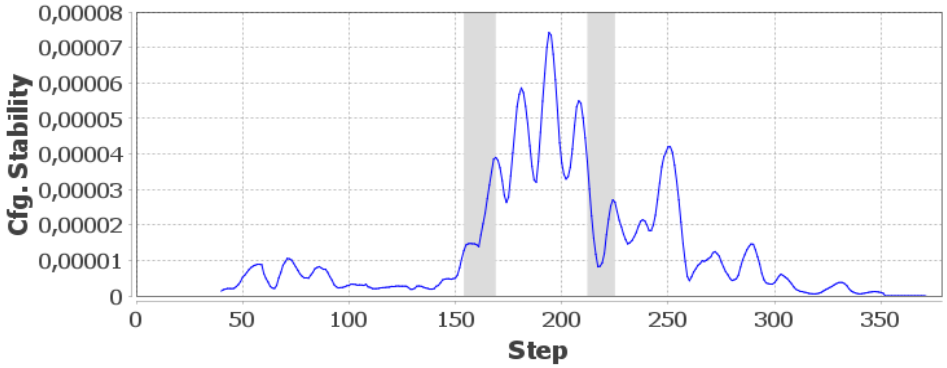


Figure 5.142: The *Configuration Stability* in the Game of Life simulation with $M = L = 20$ and $\varepsilon = 0.05$. The phases of interest are marked grey.

Configuration Variability

The Configuration Variability, as seen in Figure 5.143, again resembles the number of active cells. If the cluster count is $k \geq 2$, there will be a cluster containing all the 1s and another with all the 0s. In this case, the distance to the centre of the cluster is 0 for all cluster members. Therefore, only the summand for $k = 1$ contributes to the value of the metric.

Let a denote the number of active agents. For $k = 1$ the centre C of mass for $n - a$ zeros and a ones is $C = a/n$. The sum of distances s to the centre is

$$s = (n - a)\,\frac{a}{n} + a\left(1 - \frac{a}{n}\right) = 2a\left(1 - \frac{a}{n}\right). \tag{5.8}$$

With the given values of a, the factor $1 - a/n$ ranges from ≈ 0.8 to ≈ 0.95. Thus, the average distance $d = s/n$ is nearly proportional to a. This leads to the similarity with the number of active cells.

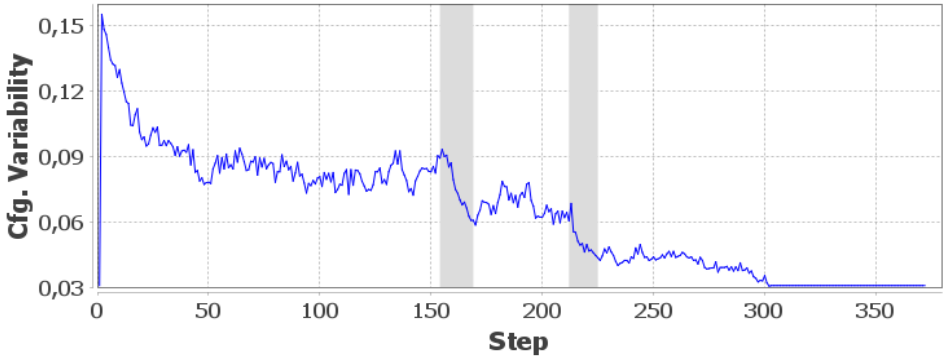


Figure 5.143: The time series of the *Configuration Variability* in the Game of Life scenario. The phases of interest are marked grey.

Coherence Metrics

Figure 5.144 shows the *Configuration Coherence*. At first glance, it looks like the inverted graph of the number of active cells, but actually, the result for a active cells is

$$coh_c = \frac{1}{1 + \frac{a}{n}\left(1 - \frac{a}{n}\right)} = \frac{1}{1 + \frac{a}{n} - \frac{a^2}{n^2}}. \tag{5.9}$$

For sufficiently small values of a, the term a^2/n^2 becomes neglectable. Thus, in our case we get:

$$coh_c \approx \frac{1}{1 + \frac{a}{n}}. \tag{5.10}$$

That explains the reciprocal similarity to the number of active cells. Nevertheless, the

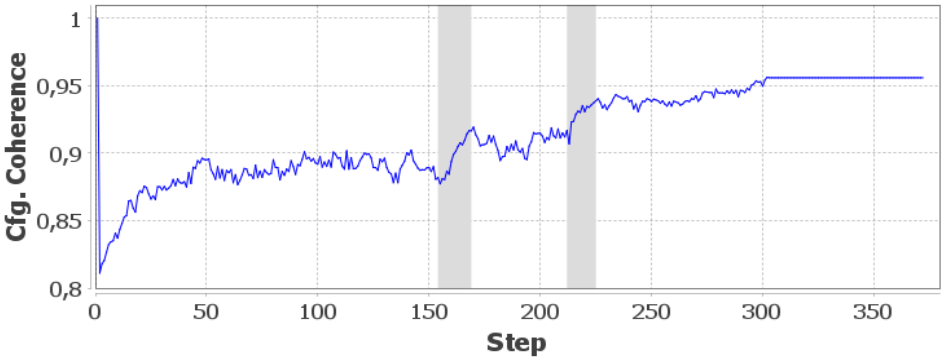


Figure 5.144: The time series of the *Configuration Coherence* in the Game of Life scenario. The phases of interest are marked grey.

two events are visible in the graph in the same way as they are visible in the number of active cells.

The *Adaptation Coherence*, on the other hand, shows only the event at $t = 212$, as shown in Figure 5.145. Here, the value reaches $coh_a \approx 0.69$. In this simulation, the *Adaptation Coherence* has values near 1.0 for the first three steps and then drops to a baseline of $coh_a \approx 0.667$. For a better overview, we cut off the Y-axis.

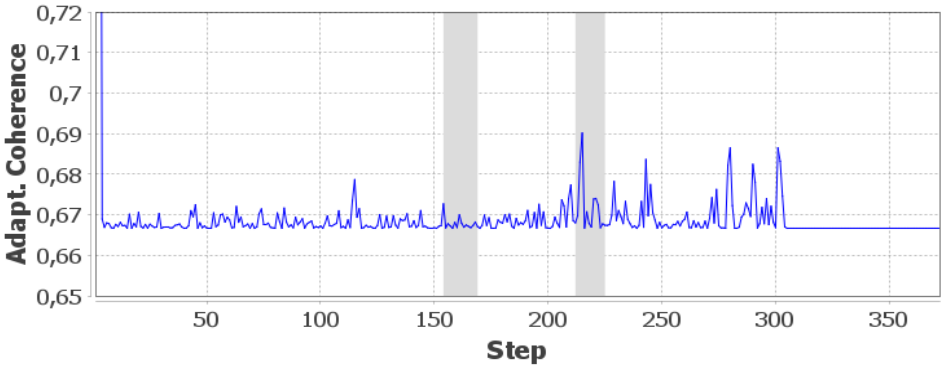


Figure 5.145: The time series of the *Adaptation Coherence* in the Game of Life scenario. The graph is zoomed in for a better overview. The phases of interest are marked grey.

Parameter Usage Metrics

The *Global Parameter Usage* U_g tells us how much of the available spectrum of values is used at each step. For the Game of Life simulation, at any given time, there are dead

and living cells present. This means the two possible configuration values are used at every step. Therefore, $U_g = 1$ for all time steps.

The *Average Parameter Usage* is shown in Figure 5.146. We see that higher values for the window parameter M flatten the graph more and lead to delays. The first event can be identified by a notable drop for all three given parameters. For $M = 5$, the drop starts at $t = 161$ from $U_a \approx 0.034$ and lasts until $t = 173$, where it reaches $U_a \approx 0.018$. For $M = 20$, the corresponding drop starts at $t = 170$ from $U_a \approx 0.056$ and ends at $t = 188$ with $U_a \approx 0.04$.

The second event can only be seen with $M = 5$. In this case, the metric creates a steep drop from $t = 216$ with $U_a \approx 0.015$ down to $U_a \approx 0.008$ at $t = 220$. For the higher values of M, this drop is lost in the general downward trend starting at $t = 200$.

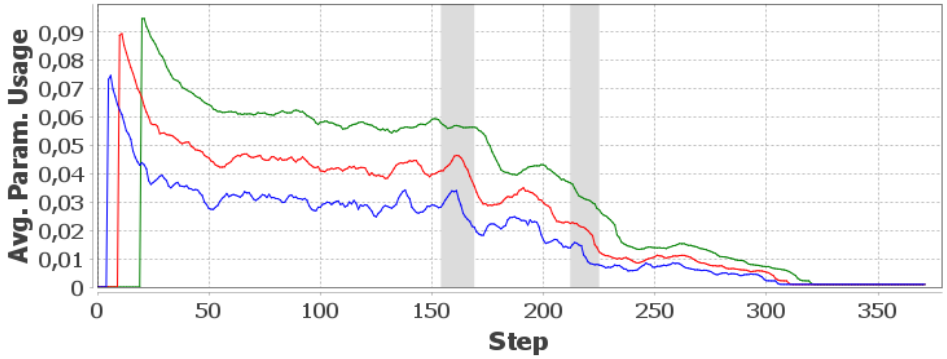


Figure 5.146: The time series of the *Average Parameter Usage* in the Game of Life scenario. The time window M is $M = 5$ (blue), $M = 10$ (red), and $M = 20$ (green). The phases of interest are marked grey.

Configuration Divergence

Finally, the *Configuration Divergence* is given in Figure 5.147. As in the other scenarios, the parameters influence the delay and the width of spikes. The two events can be identified by all given parameter combinations. $M = L = 10$ gives the best result. Both spikes are prominent and not as delayed as for $M = L = 20$. The first peak is visible at $t = 170$ with $c_d \approx 0.004$, and the second is at $t = 224$ with $c_d \approx 0.0022$. For $M = L = 5$, several additional spikes of comparable height are visible. For example, the peak at $t = 199$ has a value of $c_d \approx 0.0011$, while the peak for the second event at $t = 218$ reaches an only slightly higher level with $c_d \approx 0.018$.

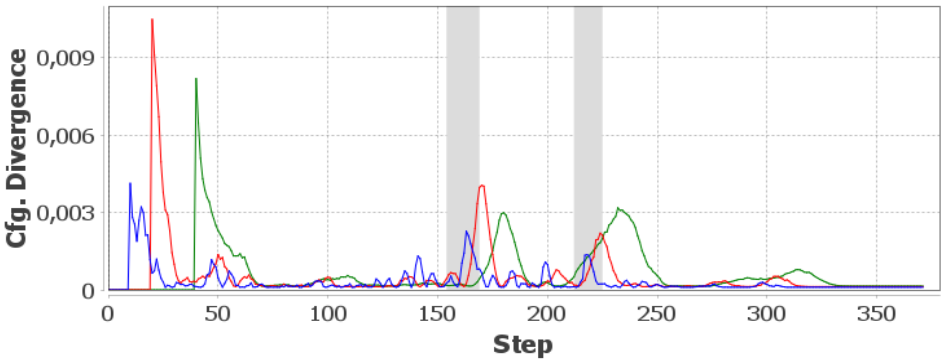


Figure 5.147: The time series of the *Configuration Divergence* in the Game of Life scenario. The parameters are $M = L = 5$ (blue), $M = L = 10$ (red), and $M = L = 20$ (green). The phases of interest are marked grey.

Self-Organisation Divergence

Again, this simulation does not implement direct communication between the cells, but with the same argument as for the Flocking scenario, we can introduce a communication that makes the *Self-Organisation Divergence* equal to the *Configuration Divergence*.

Summary

In this scenario, the *Entropy*, the *Configuration Variability*, and the *Configuration Coherence* show a strong similarity to the number of active cells. Since the number of active cells is our source for the interesting events, these three metrics are able to identify the events. Yet, they rely on the definition that the respective drop in the number of active cells is a specific event. The same holds for the *Average Parameter Usage*, with the restriction to a suitable window size. The *Configuration Stability* and the *Configuration Divergence* can identify both events by giving isolated peaks, but again only with suitable parameters. The *Adaptation Coherence* can only identify the second event, and the *Global Parameter Usage* is a flat line and thus gives no information.

5.2.11 Evaluation of Multivariate Configurations

As mentioned in Chapter 4.3.4, configurations with higher dimensions can lead to a more sparse distribution of observations and less accurate results. To illustrate this problem, we apply the multivariate version of the *Configuration Stability* to the Road Block scenario. As we can see in Figure 5.148, we lose information. Only the univariate metric shows notable activity after the end of the blockade.

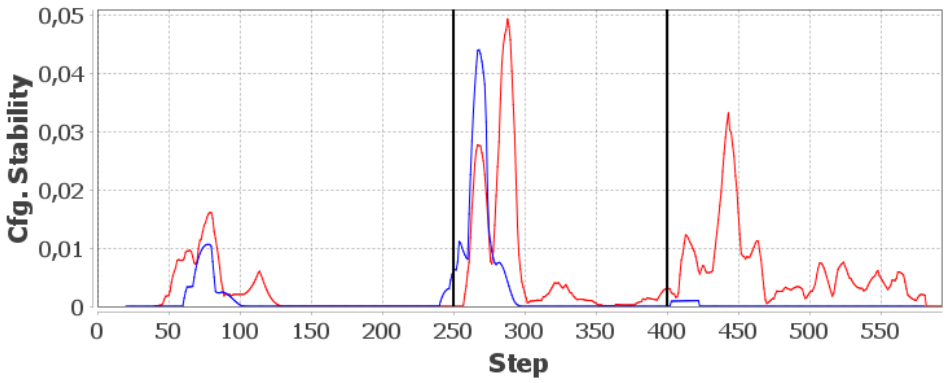


Figure 5.148: The multivariate (blue) version of the *Configuration Stability* and the univariate (red) version in the Road Block scenario. Both with $M = L = 20$ and $\varepsilon = 10$. The black lines denote the start and the end of the road block.

So, wherever possible, the analysed entries in the configuration vectors should be separated or divided into reasonable groups.

5.3 Evaluation Summary

After having applied the metrics to our selected systems and given the relevant events (i.e. the disturbances and the events of specific interest), two questions need to be answered:

1. Does the metric show a reaction to the relevant event?
2. Is the metric able to isolate this event?

For the first question, we say that the metric shows a reaction if it creates an isolated peak that can be associated with the event or if there is a change in the metric's baseline at a point in time that can be associated with the event. We collected the answers to this question in Table 5.6. Here, a green tick (✓) means that the metric shows a reaction to at least one relevant event in the scenario. If the metric fails to show a reaction or if peaks or changes in the baseline cannot be clearly linked to the events, then we use a red X (✗). For example, we see that the *Configuration Stability* is sensitive to the relevant events in all scenarios while the *Adaptation Coherence* shows a reaction in only half of the applications.

For the second question, we say that the metric can isolate the event if the metric generates an isolated peak or a substantial and unique change in the time series or if the comparison to a reference time series (e.g. the undisturbed simulation of the scenario) shows a notable difference. That means that the adaptation effects of the disturbances can be separated from the effects of other situations in the scenarios. The answers to the second question are compiled in Table 5.7. If the metric's reaction is isolated or the

difference to the reference time series is obvious, then we will use a green tick (✓). If the metric fails to isolate all events in the scenario, e.g. because it generates additional spikes or the comparison to reference time series shows no substantial difference, then we mark the metric with a red X (✗). If the metric can isolate some but not all events, we use a black cross (✤). As an example in the table, we see again that the *Configuration Stability* shows the best performance with only one scenario where it fails to isolate the event and five events where it shows all relevant events. The *Entropy*, on the other hand, fails in six of the eight scenarios.

	Scenario							
Metric	**Flocking**	**Smart Cameras**	**Traffic: Roadblock**	**Traffic: Rush Hour**	**Traffic: Life-Like**	**Traffic: Life-Like DPSS**	**Maritime Traffic**	**Game of Life**
Entropy	✓	✓	✓	✓	✗	✗	✗	✓
Cfg. Stability	✓	✓	✓	✓	✓	✓	✓	✓
Cfg. Variability	✓	✓	✓	✓	✓	✓	✗	✓
Cfg. Coherence	✓	✗	✓	✓	✗	✗	✓	✓
Adapt. Coherence	✓	✗	✓	✓	✗	✗	✗	✓
Glob. Param. Usage	✓	✗	✓	✓	✓	✗	✓	✗
Avg. Param. Usage	✓	✓	✓	✓	✗	✗	✓	✓
Cfg. Divergence	✓	✓	✓	✓	✗	✓	✗	✓
SO Divergence	✓	✓	-	-	-	✗	-	✓

Does the metric show a reaction to the relevant event?
✓ = yes, ✗ = no, - = not applicable

Table 5.6: Overview of the detection performance of the metrics in the different scenarios regarding the question: Has the relevant event a notable influence on the metric?

We will end this chapter with some remarks on the scenarios and the disturbances, a summary of each metric and a conclusion.

5.3.1 Remarks on the Scenarios

When we look at the results, we see that the overall detection performance is poor in the scenarios with the open systems, i.e. the *Life-Like Road Network* and the *Maritime Traffic* scenarios. In the *Life-Like Road Network*, scenario only three of the eight metrics are influenced by the relevant events, and isolation of the events is, at most, only partly

Metric	Scenario: Flocking	Smart Cameras	Traffic: Roadblock	Traffic: Rush Hour	Traffic: Life-Like	Traffic: Life-Like DPSS	Maritime Traffic	Game of Life
Entropy	✓	✓	✤	✤	✗	✗	✗	✗
Cfg. Stability	✓	✗	✓	✓	✤	✤	✓	✓
Cfg. Variability	✗	✓	✤	✓	✗	✤	✗	✗
Cfg. Coherence	✓	✗	✤	✓	✗	✗	✓	✗
Adapt. Coherence	✓	✗	✤	✓	✗	✗	✗	✤
Glob. Param. Usage	✓	✗	✗	✓	✤	✗	✗	✗
Avg. Param. Usage	✓	✓	✓	✓	✗	✗	✗	✗
Cfg. Divergence	✓	✓	✤	✗	✗	✤	✗	✓
SO Divergence	✓	✓	-	-	-	✗	-	✓

Is the metric able to isolate this event?
✓ = yes, ✗ = no, ✤ = partly, - = not applicable

Table 5.7: Overview of the detection performance of the metrics in the different scenarios regarding the question: Can the metric distinguish the relevant event from other events?

possible. The *Maritime Traffic* scenario allows only slightly better performance, with four metrics being influenced and only two metrics that are able to isolate the accident. In the closed systems, the metrics show better performance. One major difference between the two classes of systems in our evaluation is the frequency of configuration changes. While our closed systems change their configuration at nearly every time step, the open systems apply changes way less often. Since the metrics aim to analyse configuration changes, we can assume that this is one important reason for this observation.

5.3.2 Remarks on the Disturbances

Chapters 5.2.2 and 5.2.6 presented the influence of different strengths of the disturbance. The results show that we can make no general statement about how a change in the disturbance affects the metrics. In some cases, smaller disturbances lead to decreased reactions of the metrics and even the reaction being lost in the background noise. Here, changing the parameters of the metrics can sometimes mitigate the weakness of the reaction. In other cases, the differences between two disturbances lead to a negligible difference in the metrics. This leads to the following conclusions:

1. The choice of parameters for the metrics is crucial.
2. The definition of *disturbance* needs special attention in each system. In the Rush Hour scenario, the smallest analysed disturbance with only 100 additional cars during the rush hour might be considered only a phase with increased system load but not a disturbance as such. This definition is up to the user of the framework.
3. Using an appropriate definition of *disturbance*, when the framework is used with all applicable metrics, then a disturbance can be identified in our scenarios, regardless of the strength of the disturbance.

5.3.3 Summary for the Metrics

Tables 5.6 and 5.7 show that every metric shows a reaction to the relevant events in at least four of the eight scenarios, with only the *Adaptation Coherence* hitting this lower bound. Furthermore, every metric has at least two scenarios where it can fully isolate the relevant events.

Entropy

The *Entropy* shows a reaction to the relevant events in five of the eight scenarios. In the open systems, the influence of the relevant configuration changes shows no specific change in the behaviour of the metrics' time series. In the Rush Hour and the Game of Life Scenario, the reaction is relatively small. In the other scenarios, the reaction is substantial.

The ability to isolate the relevant events is only given in the Flocking and the Smart Camera scenarios. In the Rush Hour and the Roadblock scenario, only one of the two events is clearly visible in the time series.

Configuration Stability

The *Configuration Stability* is sensitive to at least one relevant event in all scenarios. The evaluations show that the sensitivity heavily depends on the choice of parameters. The double window approach in this metric often leads to delays in the reaction of the time series. The ability to isolate an event can suffer from this delay, as we can see in the Smart Camera scenario. The *Configuration Stability* is one of the few metrics that can isolate two of the three relevant events in the Life-Like Road Network scenarios. Only the second road block cannot be isolated here.

In general, we can state that this metric shows the best performance of all presented metrics. With a sensitivity in all scenarios and only one scenario where the metric fails to isolate any event, the *Configuration Stability* takes the first place.

Configuration Variability

When it comes to sensitivity, the *Configuration Variability* places second with only the Maritime Traffic scenario, where the configuration changes cannot be clearly said to influence the time series. In the Life-Like Road Network scenarios, the influence of at least one relevant event is given. In the OTC-only simulation, this influence can only be seen in the direct comparison with the undisturbed case.

The isolation of events is fully given only in two scenarios and at least partly in two others, including one of the Life-Like Road Network scenarios.

Configuration Coherence

The *Configuration Coherence* shows a sensitivity in five of the eight scenarios. An event isolation is possible in four of them, including the Road Block scenario where only one of the two events can be isolated. This metric is one of the two metrics that allow the isolation of the event in the Maritime Traffic scenario, although only through comparison with the undisturbed case.

Adaptation Coherence

In the Flocking scenario, the Rush Hour scenario, and the Road Block scenario, the *Adaptation Coherence* appears like a noisier version of the *Configuration Coherence*. However, the approach to only consider those subsystems that had a configuration change since the last time step shows a different behaviour in the other five applications. Here, the time series of the two coherence metrics have nothing in common.

The *Adaptation Coherence* shows sensitivity in only half of the scenarios but can, in all of these, at least partly isolate an event. Especially in the open systems, this metric shows poor performance.

Global Parameter Usage

The *Global Parameter Usage* shows an event reaction in five of the eight scenarios. With five scenarios where it fails to isolate any event, this metric is the weakest in our scenarios. Yet, the relatively coarse approach to simply monitor the fraction of used parameters can fully isolate the relevant events in the Flocking scenario and the Rush Hour scenario. Even the OTC-only version of the Life-Like Road Network scenario allows the isolation of at least one event. This scenario is the one where the least number of metrics are able to isolate an event. The *Global Parameter Usage* is one of the two metrics that can do so.

Average Parameter Usage

The *Average Parameter Usage* is affected by at least one relevant event in six of the eight scenarios. The isolation of these events is possible in four of the scenarios. The Game of Life scenario and, again, the open systems do not allow the event isolation, no matter what parameter is used for the metric.

Configuration Divergence

This metric gives a reaction to an event in six of eight scenarios, and in five of them, at least one event can be isolated. This includes the DPSS version of the Life-Like Road Network, where most metrics fail to isolate any event.

Self-Organisation Divergence

Due to its nature, this metric is fully applicable only to the DPSS version of the Life-Like Road Network scenario. In this case, it fails to react to an event or even isolate it. This comes from the low number of messages in the scenario. In three other scenarios, we emulated a message exchange so that the *Self-Organisation Divergence* became equal to the *Configuration Divergence*. In these scenarios, the *Configuration Divergence* - and therefore the *Self-Organisation Divergence* - can isolate the relevant events.

5.3.4 Conclusion

The evaluation shows that all metrics can give an indication of unusual events in at least two different scenarios. However, they cannot be used alone. No metric is able to isolate the events in all given scenarios. Only in combination with other metrics can the events be (at least partly) identified. Therefore, our idea of an overarching framework makes sense, and all of our metrics are eligible to be integrated into this framework.

Chapter 6

An Application towards Self-Explanation

In the previous chapter, we saw how our framework is able to give indicators for unusual self-adaptation events in a SASO system. The created time series for themselves show no value for a simple user. Only a system administrator or an automated monitoring system can benefit from the framework's output.

In the introduction in Chapter 1.2, we noted that a user will accept an unusual behaviour of an SASO system if it provides an explanation for the observed behaviour. This chapter shows how our framework can help to provide such self-explanation capabilities to SASO systems. Besides having an indication of the presence of an unusual event, the source of this event is often needed to give a useful explanation.

Therefore, we will show how the framework can be extended to aid in a root cause analysis of such an event. For this, we first give a short introduction to the notion of root cause analysis in Chapter 6.1. Then, Chapter 6.2 shows the necessary extensions to the framework. With these modifications at hand, Chapter 6.3 shows the actual application to three of our scenarios. After an overview of other approaches to root cause analysis (Chapter 6.4) and a comparison to a classical method (Chapter 6.5), we conclude the use case with the summary in Chapter 6.6.

6.1 Root Cause Analysis

In general, the notion of *root cause analysis* (RCA) refers to a structured process of problem-solving used to identify the initial causes of errors or problems [245]. In Chapter 1.2, we used a traffic light as an example where the user experiences an unusually long phase of red light. In this example, the technical root cause for the problem of

the long red phase is the car accident that happened somewhere in the street network. From another point of view, the root cause lies deeper: a malfunction in the car, a medical problem of the driver or something completely different. Eventually, the root cause depends on the definition of the problem and the person who defines it.

RCA is used in productive environments in many domains, such as healthcare [168], manufacturing [130], telecommunication [254] and IT operations [212]. Depending on the domain, the problem and the available tools, the actual methods to perform an RCA can differ significantly. During the Covid pandemic, some medical authorities were forced to use pen, paper and telephones to trace contact persons and find the root of infection chains [259]. On the other hand, in large-scale manufacturing chains, machine-learning approaches using big data sources can be applied [133].

No matter what tools are used, a complete root cause analysis comprises the following four steps [245]:

1. Description of the problem.
2. Creation of a timeline from the normal situation until the problem occurs.
3. Identification of the root cause and other causal factors.
4. Creation of a causal graph from the root cause to the problem.

If we assume that the experienced problem is the result of an unexpected or unusual behaviour of the system and use our core assumption that an unusual behaviour comes from unusual self-adaptation events, then we can apply our framework and identify the point in time where the metrics in our framework indicate such an event. Together with this temporal identification of possible root causes, the framework can give hints for a spatial identification: which subsystems are most likely linked to the root cause? The framework assists in the steps 2 and 3 of the RCA by helping to answer the questions of "Where?" and "When?".

6.2 Extension of the Measurement Framework

To aid in the spatial detection of events, the involved subsystems have to be associated with the event. To achieve this, we extend our framework to assign a score to each subsystem. Following our core assumption, we calculate the score based on configuration changes. For every metric and every subsystem, we investigate how much impact the current configuration change of the subsystem has on the value of the metric. A higher impact will lead to a higher score. Figure 6.1 shows the idea.

The calculation of the scores requires only small additions. During the computation of the time series of the metrics, every subsystem has to be queried to deliver its configuration. The association between the subsystem and the result of the metric calculation is directly available. Thus, the only additional step is to calculate the score for the

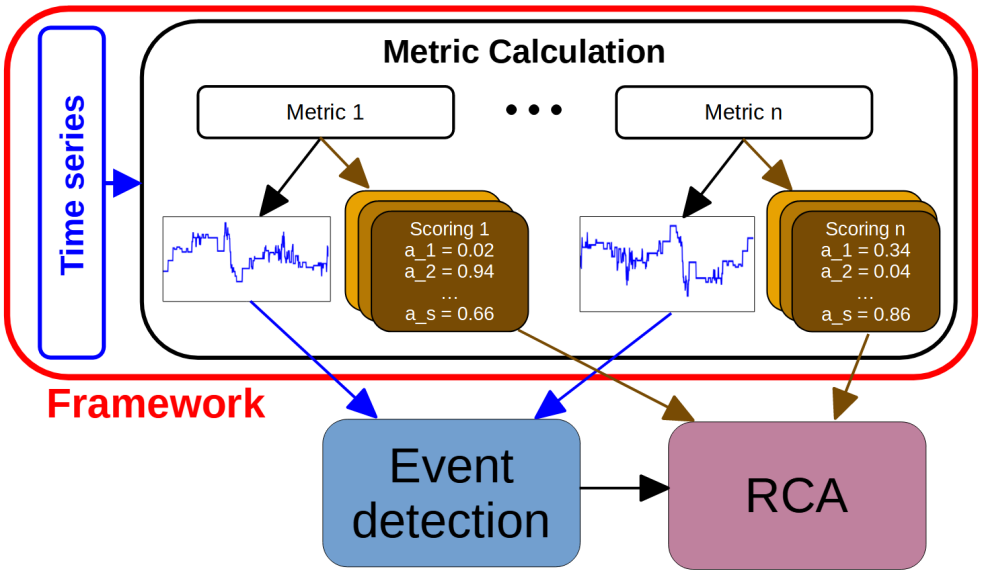


Figure 6.1: The extension of the framework. For every time step, each metric calculates a score for every subsystem. Together with the event detection in the time series, the scores are used in the RCA.

subsystem. We will see the formulae for this in Chapter 6.2.2.

The procedure to identify likely sources of an unusual event starts with the temporal detection of the events by analysing the time series that our metrics generate. Chapter 6.2.1 will give details on this. Then, when the metrics indicate an unusual event, we take a look at the sum of the scores of the subsystems. If we have only a handful of subsystems with a significantly higher score than the rest, we can assume that the source of the event is associated with that group. If, otherwise, the scores for all subsystems are on the same level, we can assume that it is a global event and the source cannot be located in isolated subsystems. Starting with the first case, we can simply sort the subsystems by their scores and start inspecting the first I subsystems with the highest scores (for a fixed number I). If we cannot identify a source in that list, we again assume a global event.

6.2.1 Formalisation of the Temporal Event Detection

During the evaluation of the scenarios in Chapter 5.2, we identified peaks and spikes with the naked eye. For an application in the real-world this would require a human being to watch the time series and have the knowledge and experience to identify the peaks. This becomes unmanageable very quickly. Since SASO systems aim to reduce the management effort, an automatic method to identify peaks is required.

For this, we apply one of the many peak detection algorithms for univariate time series [162]. We will use the algorithm presented in [234]: this algorithm takes a time series as input and creates a new time series as output that consists of 0s and 1s. A 1 indicates that the corresponding entry in the input time series is a peak. The algorithm outputs a 1 if the last data point (which is at runtime a newly added data point) is a given number x of standard deviations away from some moving mean. Otherwise, we receive a 0. The algorithm takes three inputs: the parameter "lag" is the size of the moving window, the parameter "threshold" is the z-score at which the algorithm signals and the parameter "influence" is the influence (between 0 and 1) of new signals on the mean and standard deviation. The algorithm creates signals with values $+1$ for a positive deviation (i.e. the new data point is above the moving mean) and -1 for a negative deviation. Since we are only interested in the existence of a peak signal, we will use the absolute value of the signal.

The benefit of this algorithm is its ability to be used at runtime. On the downside, suitable values for the three parameters depend on the metric and the SASO system. To find such values, experience and experiments with data from the system are required.

One important point for temporal event detection is that for a given SASO system, usually, not all metrics in our framework produce usable results. This means only the output of some selected metrics can be used for a specific system. This selection again requires experiments with the system.

After selecting the suitable metrics for our system and configuring the peak detection algorithm, we must fix a threshold n_{thres}. This is an integer and depends on the number of selected metrics. Counting the number of peak signals n_{peak} provided by the algorithm for the metrics at a given time, we finally can define that an unusual event occurs if $n_{peak} \geq n_{thres}$ holds.

6.2.2 Scoring

We now explain how a score can be derived during the calculation of the metrics. Since we have several metrics, the resulting score from every metric should be comparable. Therefore, we scale the score calculation such that the score for every metric lies in the interval $[0, 1]$. The subsystems with the highest impact on the metric are assigned the highest score. The following calculations are not the only possibilities to assign a score value to a subsystem. The user is free to adapt them as necessary. The given formulas work well for the application scenarios presented in Chapter 6.3.

Entropy

When computing the *Entropy*, the configuration value of a subsystem is put into the proper bin where it (usually) becomes one of many entries. Only the number of entries

in the bin is used for the *Entropy*. There is no evident way to quantify the influence of a single subsystem on the computation result. Thus, assigning a score is not possible, and the *Entropy* is not used in the RCA application.

Configuration Stability

During the calculation of *Configuration Stability*, we classify agents as active or inactive depending on how strong their configurations have changed over time. This classification is the basis for the computation of the metric. Therefore, we assign those agents a score of 1 if they are active by definition. The inactive agents are assigned a 0.

For a fixed point t in time and subsystem a, the score $s_{\text{cfg.stab.}}$ is defined as

$$s_{\text{cfg.stab.}}(a) = \begin{cases} 1, & \text{if } a \text{ is classified as active} \\ 0, & \text{otherwise.} \end{cases} \tag{6.1}$$

Configuration Variability

The effect of a changed configuration vector on the result of a clustering algorithm is hard to predict and even harder if we use several cluster counts. Therefore, a differentiated scoring is not reasonable. Thus, for this metric, we assign a score value of 1 to every subsystem whose configuration changed since the last time step.

Let $x_{a,t}$ and $x_{a,t-1}$ be the configuration vectors for the subsystem a at time t and $t-1$ respectively. The score $s_{\text{cfg.var.}}(a)$ for this subsystem is then

$$s_{\text{cfg.var.}}(a) = \begin{cases} 0, & \text{if } x_{a,t} = x_{a,t-1} \\ 1, & \text{otherwise.} \end{cases} \tag{6.2}$$

Coherence

The calculation of these metrics is based on the distances of the configuration vectors to the average. Therefore, we assign the score based on that distance: let $d(x_a) := (\|x_a - \hat{x}\|)$ be the distance of the configuration from the average for the subsystem a and let $m := max(d(x_a))$ be the maximum of these distances. The score $s_{\text{cfg.coh.}}$ for the subsystem x_a is then

$$s_{\text{cfg.coh.}}(a) = \frac{d(x_a)}{m}. \tag{6.3}$$

Therefore, the subsystem with the largest distance receives a score of 1. The calculation for the *Configuration Coherence* and the *Adaptation Coherence* differs only in the considered subsystems in the same way as the metrics do. The *Configuration Coherence* uses all subsystems, while the *Adaptation Coherence* uses only those with a configuration change. In the latter case, the subsystems without a change receive the value $s_{\text{cfg.coh.}} = 0$.

Global Parameter Usage

In this metric, the configuration of a subsystem either contributes to the bounds of the total range or not. A subsystem is assigned a score of 0 if its current configuration does not touch the current bounds. Those subsystems in which either the lower bound of the configuration range equals the lower bound of the total range or the upper bound of the configuration range equals the upper bound of the total range are assigned a score of 1.

For the current time frame and a fixed configuration index j, let $r_s := [r_{s,l}, r_{s,u}]$ be the current maximum interval of used values in the system with the lower bound $r_{s,l}$ and the upper bound $r_{s,u}$. Analogously, let $r_a := [r_{a,l}, r_{a,u}]$ be the interval of configuration values of the subsystem a. The score $s_{glob.param.}$ is then

$$s_{\text{glob.param.}}(a) = \begin{cases} 0, & \text{if } r_{a,u} \neq r_{s,u} \text{ and } r_{a,l} \neq r_{s,l} \\ 1, & \text{otherwise.} \end{cases} \tag{6.4}$$

Average Parameter Usage

For this metric, we are looking at how much the ranges (i.e. the length of the interval) of used values change from one time step to the next. Analogous to the scoring in the *Coherence* metrics, the subsystem with the strongest change receives a score of 1. The score for the other systems is scaled accordingly. Subsystems whose range did not change receive a score of 0.

For a fixed configuration index j, let $r_{a,t}$ and $r_{a,t-1}$ the range of values of the configuration entry $x_a[j]$ for the subsystem a in the current time frame f and the previous frame $f-1$ respectively. Let $d(x_a) := (\|r_{a,f} - r_{a,f-1}\|)$ be the difference of these ranges, and let $m := max(d(x_a))$. The score $s_{avg.param.}$ is then

$$s_{\text{avg.param.}}(a) = \begin{cases} 0, & \text{if } r_{a,f} = r_{a,f-1} \\ \frac{d(x_a)}{m}, & \text{otherwise.} \end{cases}$$

We need to note that a score of 0 can occur even if the actual configuration values have changed, e.g. when a has values in $[0, 2]$ in the time frame f and values in $[3, 5]$ in $f-1$ both value ranges are equal

Configuration Divergence

The *Configuration Divergence* uses the configuration values to calculate a probability distribution. Like in the *Entropy*, the influence of a single subsystem cannot be traced during the calculation. Therefore, we will exclude this metric from our RCA application.

Self-Organisation Divergence

This is another metric where a scoring cannot be derived. The reason is that we usually cannot associate a subsystem with the result of a message because there is an ambiguity between the sender and the receiver. We cannot know whether the message is a result of the event at the sender or the cause of an event at the receiver. Additionally, the same arguments as for the *Configuration Divergence* apply here since the *Self-Organisation Divergence* uses probability distributions.

6.3 Application of the Measurement Framework

In the set of scenarios introduced in Chapter 5.1, there are three where we can put our finger on a single source of disturbance and where most of our metrics show a sufficient performance:

1. The Flocking scenario (cf. Chapter 5.1.1) with the shot at a single bird.
2. The Smart Camera scenario (cf. Chapter 5.1.2) where the attacker influences the camera.
3. The Artificial Road Block scenario (cf. Chapter 5.1.3) where a single intersection becomes unavailable.

We will now show how our framework with its scoring extension works in these three applications.

6.3.1 Application in the Flocking Scenario

In the Flocking scenario, we installed a disturbance at time step $t = 500$, which caused several birds to change their direction sharply. To create a spatial identification of the cause, we will first set up the framework with suitable parameters, then we will show the output of the framework and finally analyse the results.

Configuration of the Framework

For this application, we choose the *Configuration Coherence*, the *Configuration Variability*, and the *Average Parameter Usage* as sources for the scores and the peak signals. As the evaluation results in Chapter 5.2.1 show, these three metrics have a fast reaction to the disturbance and a different behaviour in their time series.

The window size Parameter L in the *Average Parameter Usage* is set to $L = 5$. The two other metrics are without parameters. The peak detection algorithm is configured with the parameters $lag = 30$, $threshold = 5.0$ and $influence = 1.0$.

Framework Output

Figures 6.2, 6.3 and 6.4 show the time series of the three metrics and the peak signals they produce. For a better overview, the time series of the peak signals are scaled to a range that allows it to be put beside the metric. The actual signal has either a value of 0 or 1.

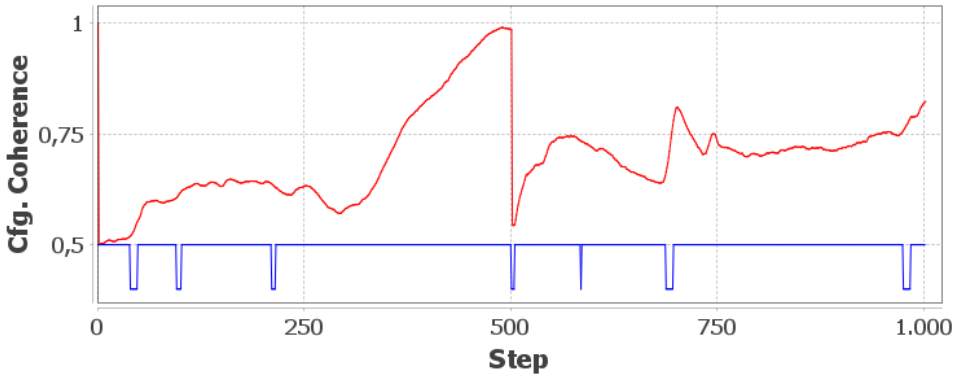


Figure 6.2: The output of the *Configuration Coherence* and its peak signals for Flocking scenario. The time series of the metrics is in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The disturbance happens at $t = 500$.

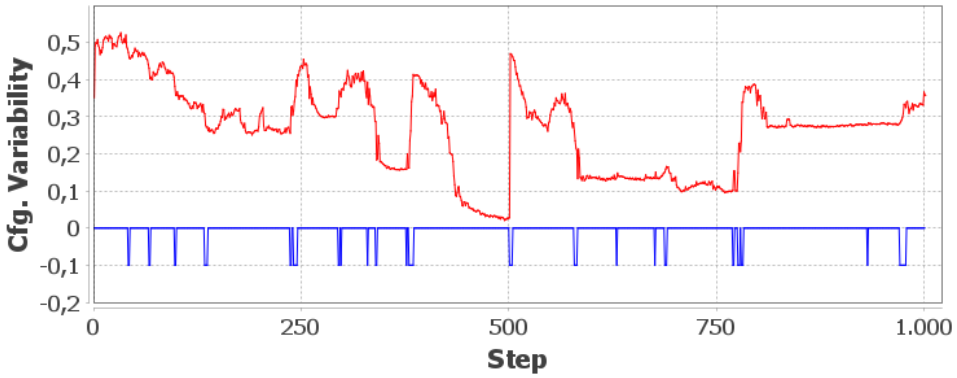


Figure 6.3: The output of the *Configuration Variability* and its peak signals for Flocking scenario. The time series of the metric is shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The disturbance happens at $t = 500$.

There are three time frames where all three metrics create a peak signal. The peak signal sum equals 3 from $t = 502$ to $t = 504$, from $t = 698$ to $t = 690$, and from $t = 975$ to $t = 976$. Our attention is on the first frame because the direct reaction to the disturbance lies in this frame. The *Configuration Variability* assigns only 1s and 0s as scores, which alone does not provide further insight. Since we are interested in the

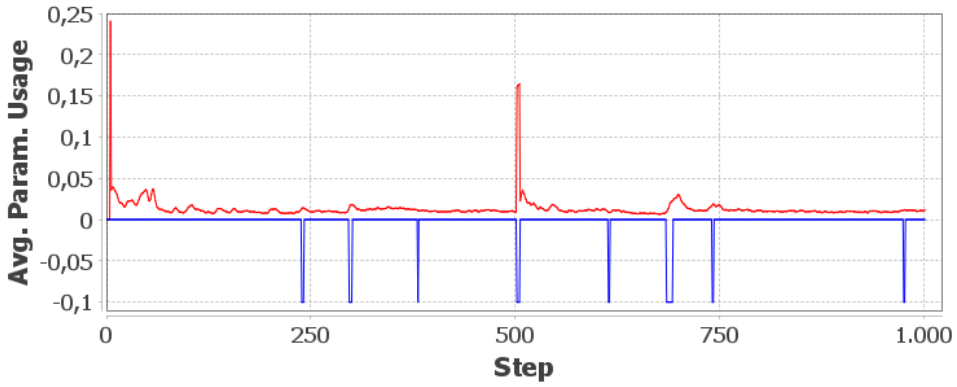


Figure 6.4: The output of the *Average Parameter Usage* and its peak signals for Flocking scenario. The time series of the metric is shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The disturbance happens at $t = 500$.

top-scoring subsystem, we provide the five highest-scoring birds for the other two metrics in Tables 6.1 and 6.2.

Step	**Subsystem scores**				
502	$a_{41} = 0.95$	$a_{39} = 1.00$	$a_{12} = 0.92$	$a_{11} = 1.00$	$a_{8} = 0.98$
503	$a_{40} = 0.34$	$a_{27} = 0.64$	$a_{26} = 0.47$	$a_{19} = 1.00$	$a_{18} = 0.50$
504	$a_{36} = 0.72$	$a_{34} = 0.67$	$a_{25} = 0.70$	$a_{15} = 1.00$	$a_{7} = 0.67$

Table 6.1: List of the highest-scoring subsystems in the *Average Parameter Usage* metric in the Flocking scenario shortly after the disturbance.

Step	**Subsystem scores**				
502	$a_{27} = 0.41$	$a_{26} = 0.39$	$a_{19} = 1.00$	$a_{11} = 0.38$	$a_{8} = 0.40$
503	$a_{43} = 0.59$	$a_{38} = 0.87$	$a_{26} = 0.73$	$a_{14} = 0.87$	$a_{13} = 1.00$
504	$a_{31} = 1.00$	$a_{27} = 0.48$	$a_{11} = 0.43$	$a_{10} = 0.74$	$a_{2} = 0.59$

Table 6.2: List of the highest-scoring birds in the *Configuration Coherence* metric in the Flocking scenario shortly after the disturbance.

Analysis

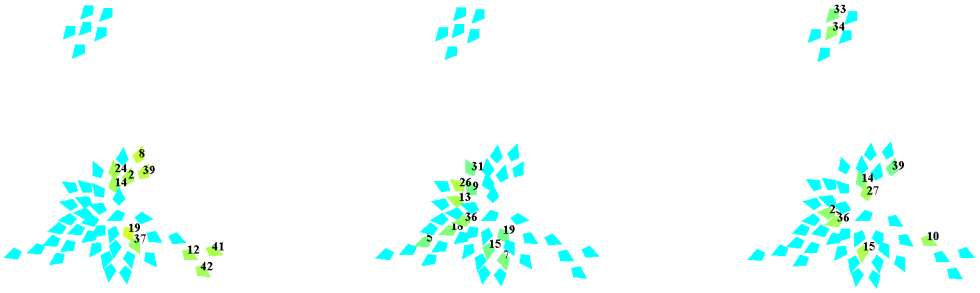
For the analysis, the first relevant information is the sum of the scores for the subsystems. Table 6.3 shows the ten birds with the highest score for the three time steps.

We see that a bird appears in the top ten only for one or two time steps. This alone seems to be no indication towards the source of the event. However, when we take the second relevant information – the locations of the birds – into account, we obtain a better

Step	Subsystem score sum				
502	$a_{42} = 1.98$	$a_{41} = 2.05$	$a_{39} = 2.28$	$a_{37} = 1.85$	$a_{24} = 2.04$
	$a_{19} = 2.57$	$a_{14} = 1.96$	$a_{12} = 1.99$	$a_8 = 2.38$	$a_2 = 2.07$
503	$a_{36} = 1.80$	$a_{31} = 1.36$	$a_{26} = 2.20$	$a_{19} = 1.36$	$a_{18} = 1.59$
	$a_{15} = 1.38$	$a_{13} = 2.00$	$a_9 = 1.34$	$a_7 = 1.53$	$a_5 = 1.40$
504	$a_{47} = 1.32$	$a_{39} = 1.37$	$a_{36} = 1.94$	$a_{34} = 1.74$	$a_{33} = 1.51$
	$a_{27} = 1.95$	$a_{25} = 1.74$	$a_{15} = 2.07$	$a_{14} = 1.51$	$a_{10} = 1.88$

Table 6.3: List of the highest-scoring subsystems in the Flocking scenario shortly after the disturbance.

view. Figure 6.5 shows the simulation during the first relevant time frame. The ten birds with the highest score are highlighted.



(a) $t = 502$ (b) $t = 503$ (c) $t = 504$

Figure 6.5: The 10 highest-scoring birds in the Flocking scenario shortly after the disturbance.

With these pictures and human expertise, we can identify the location of the source event. One possibility to achieve this is to extrapolate the current course backwards by drawing a straight line through the highest-scoring birds. Most of these lines will cross in a single point for all three time steps. And that point is the actual point in space where the shot was located. Furthermore, we can see that the number of highest-scoring subsystems considered in the analysis should not be too high. The birds a_{34} and a_{33} are on places 6 and 7 in the top 10 list but were not affected by the shot. A restriction to the top 5 would have given an even more concise picture.

6.3.2 Application in the Smart Camera Scenario

In the Smart Camera scenario, the artificial disturbance occurs between $t = 110$ and $t = 120$. During this time frame, some cameras are turned around by an external hacker. To identify the physical source of the anomaly in the system, we will first set up the

framework with suitable parameters, then we will show the output of the framework and finally analyse the results.

Configuration of the Framework

The evaluation of this scenario (see Chapter 5.2.3) shows that several metrics produce comparable time series. The *Configuration Variability* and the *Entropy* show similar behaviour. The same holds for the *Global Parameter Usage* and the *Configuration Coherence*, and the *Adaptation Coherence* and the *Configuration Divergence* as another pair. Therefore, we again configure the framework to use metrics with different behaviour. In this case, we choose the *Configuration Variability*, the *Average Parameter Usage*, and the *Adaptation Coherence*.

	Parameters	
	Metric	**Peak Detection**
Average Parameter Usage	$L = 5$	$threshold = 4.0$
Configuration Variability	-	$threshold = 25.0$
Adaptation Coherence	-	$threshold = 3.0$

Table 6.4: Configuration parameters of the framework for the Smart Camera scenario. In all three metrics, the peak detection uses $lag = 5$ and $influence = 1.0$.

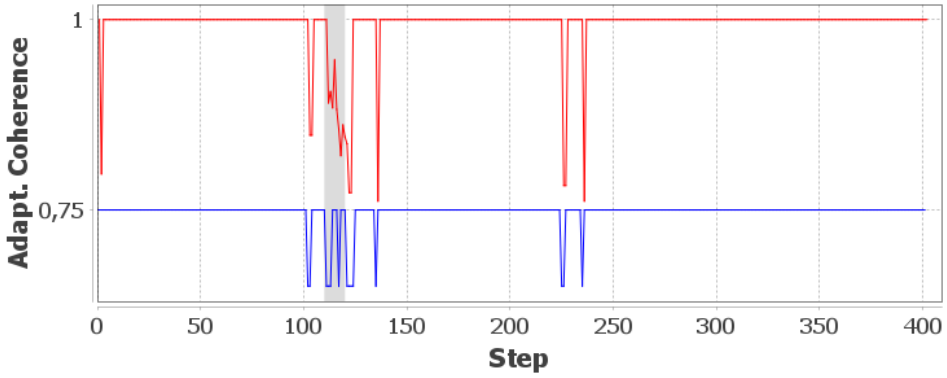
Framework Output

Figure 6.6 shows the time series of the three metrics and the peak signals they produce. For a better overview, the time series of the peak signals are scaled to a range that allows it to be put beside the metric. Of course, the actual signal has either a value of 0 or 1.

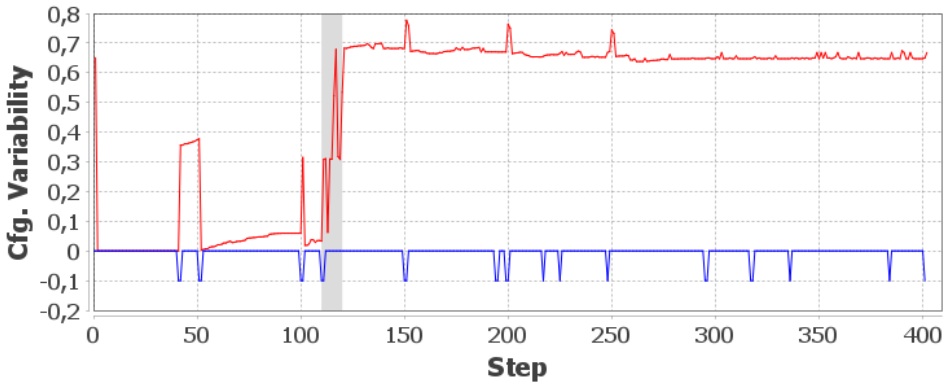
There is exactly one point in time where all three metrics create a peak signal: $t = 111$, one time step after the beginning of the disturbance. Table 6.5 lists the scores for the cameras.

Metric	**Scores**
Average Parameter Usage	$a_5 = 1.00$, $a_6 = 0.34$, $a_8 = 0.34$, $a_7 = 0.34$, $a_1 = 0.08$
Configuration Variability	$a_1 = 1.00$, $a_5 = 1.00$, $a_6 = 1.00$, $a_7 = 1.00$, $a_8 = 1.00$
Adaptation Coherence	$a_5 = 1.00$, $a_7 = 0.34$, $a_8 = 0.33$, $a_6 = 0.33$, $a_6 = 0.07$

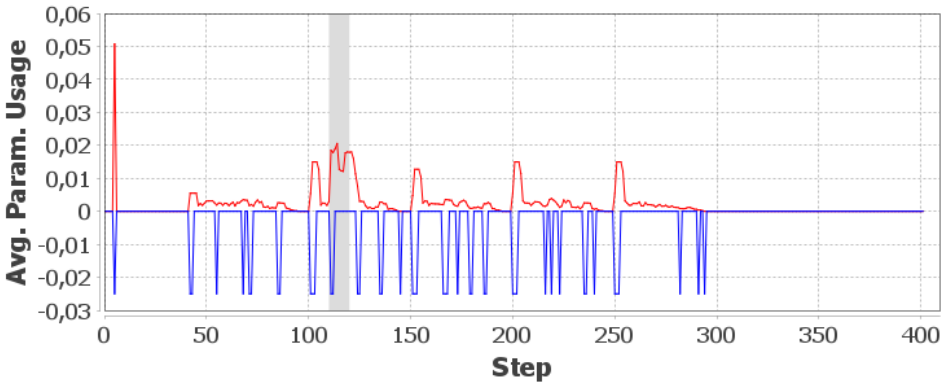
Table 6.5: Score results from the metrics in the Smart Camera scenario at $t = 111$. Cameras without a positive score are omitted.



(a) The Adaptation Coherence



(b) The Configuration Variability



(c) The Average Parameter Usage

Figure 6.6: The output of the metrics and their signals for the Smart Camera scenario. The time series of the metrics are shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The duration of the attack is marked grey.

Analysis

Figure 6.7 shows the camera positions and their assigned scores. As we can see, camera a_5 has the highest score. In fact, this is the camera where the intruder had to apply the biggest change so that the hallway on the left is free. This happened in the previous step, and the difference is recognised by the metrics. This camera does not move in the following steps. On the other hand, cameras a_6, a_7, and a_8 continuously need small adjustments to cancel out the impact of a_5 and a_1 in the monitoring system of the museum. The score for camera a_1 is the result of the moving guard which the camera currently follows.

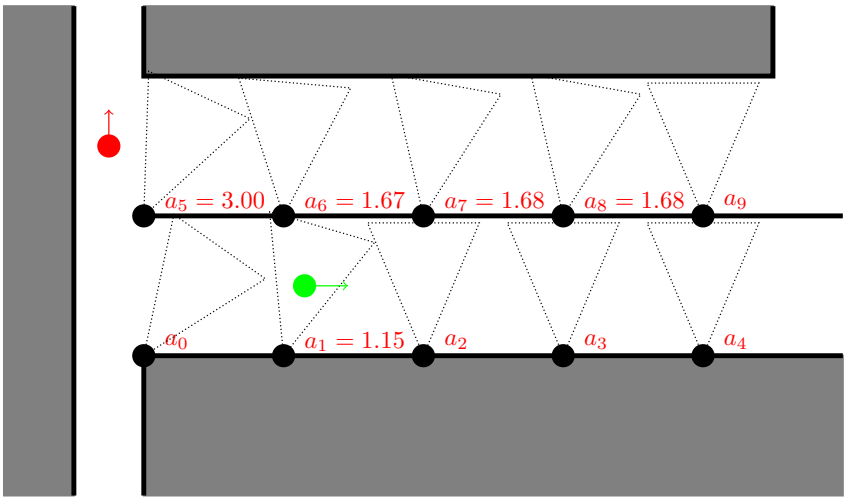


Figure 6.7: The hacked Smart Camera simulation at $t = 111$ with the resulting scores. Only the positive scores are shown.

This analysis points to the source of events within the boundaries of the camera system. The deeper cause – the external actions of the intruder and its presence – can only be recognised with additional steps, such as an alert to the guard, who might turn back and look at the hallway and the camera a_5.

6.3.3 Application in the Artificial Road Block Scenario

In the artificial Road Block scenario, our attention lies on the start ($t = 250$) and the end ($t = 400$) of the road block. From the evaluation in Chapter 5.2.4, we already know that our metrics have difficulties in identifying the end of the blockade. Therefore, we will investigate the framework output for $t \geq 250$. For this, we will first set up the framework with suitable parameters, then we will show the output of the framework and finally analyse the results.

Configuration of the Framework

With the same arguments as in the previous analysis scenarios, our choice of metrics falls on the *Configuration Variability*, the *Average Parameter Usage*, and the *Configuration Coherence*. Their parameters are shown in Table 6.6.

	Parameters	
	Metric	**Peak Detection**
Average Parameter Usage	$L = 5$	$threshold = 3.33$, $influence = 0.1$
Configuration Variability	-	$threshold = 4.0$, $influence = 1.0$
Configuration Coherence	-	$threshold = 3.33$, $influence = 0.1$

Table 6.6: Configuration parameters of the framework for the Road Block scenario. In all three metrics, the peak detection uses $lag = 15$.

Framework Output

In Figures 6.8, 6.9 and 6.10, we see the time series of the three metrics and their peak signals. Again, for a better side-by-side comparison, we moved and scaled the graphs of the peak signals. As usual, the actual signal has either a value of 0 or 1.

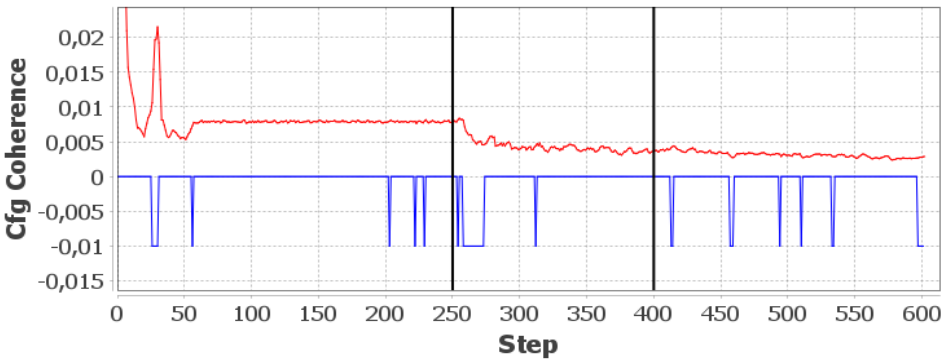


Figure 6.8: The output of the *Configuration Coherence* and its peak signals for the Road Block scenario. The time series of the metric is shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The black lines denote the start and the end of the road block.

This time, the only time frame where the peak signals sum up to 3 is from $t = 258$ to $t = 259$. Tables 6.7, 6.8, and 6.9 show the ten highest-scoring subsystems (i.e. the individual red lights) in the system for the applied metrics. Not all of the subsystems receive a positive score. In this case, they are not listed in the tables.

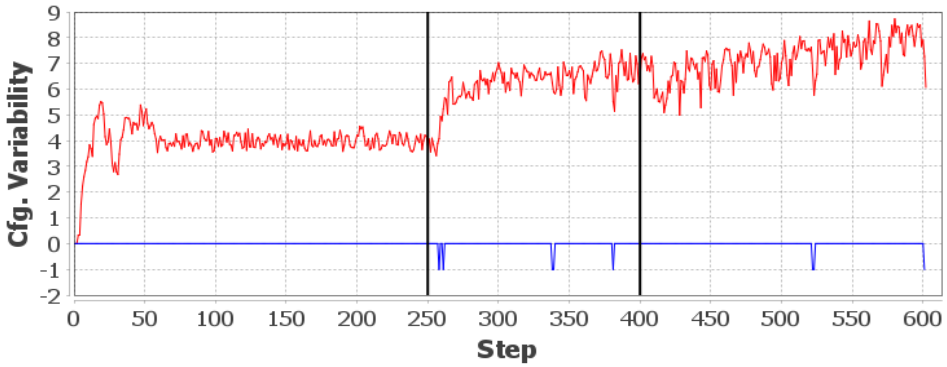


Figure 6.9: The output of the *Configuration Variability* and its peak signals for the Road Block scenario. The time series of the metric is shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The black lines denote the start and the end of the road block.

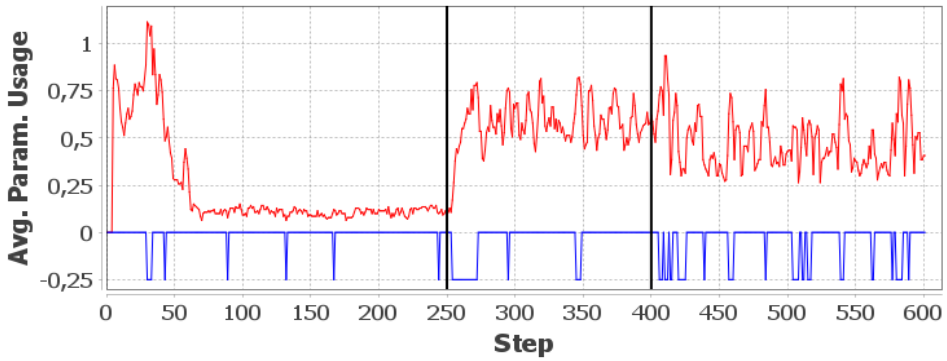


Figure 6.10: The output of the *Average Parameter Usage* and its peak signals for the Road Block scenario. The time series of the metric is shown in red, the peak signals are in blue. The signals are scaled and moved to fit in the picture. The black lines denote the start and the end of the road block.

Step	**Score for the *Configuration Coherence***				
258	$r_1 = 1.00$	$r_3 = 0.49$	$r_6 = 0.38$	$r_{12} = 0.38$	$r_0 = 0.22$
	$r_2 = 0.22$	$r_4 = 0.22$	$r_9 = 0.20$	$r_7 = 0.16$	$r_5 = 0.12$
259	$r_1 = 1.00$	$r_3 = 0.77$	$r_0 = 0.43$	$r_2 = 0.43$	$r_4 = 0.43$
	$r_6 = 0.42$	$r_{10} = 0.27$	$r_5 = 0.27$	$r_9 = 0.24$	$r_{11} = 0.17$

Table 6.7: List of the highest-scoring subsystems in the *Configuration Coherence* metric in the Road Block scenario shortly after the first disturbance.

Step	Score for the *Average Parameter Usage*				
258	$r_5 = 1.00$	$r_8 = 1.00$	$r_{13} = 1.00$	$r_4 = 1.00$	$r_{10} = 1.00$
259	$r_8 = 1.00$	$r_1 = 1.00$	$r_7 = 1.00$	$r_{13} = 1.00$	$r_{10} = 0.67$
	$r_{12} = 0.33$				

Table 6.8: List of the highest-scoring subsystems in the *Average Parameter Usage* metric in the Road Block scenario shortly after the first disturbance.

Step	Score for the *Configuration Variability*				
258	$r_1 = 1.00$	$r_6 = 1.00$	$r_7 = 1.00$	$r_8 = 1.00$	$r_9 = 1.00$
	$r_{10} = 1.00$	$r_{11} = 1.00$	$r_{12} = 1.00$	$r_{13} = 1.00$	
259	$r_1 = 1.00$	$r_6 = 1.00$	$r_7 = 1.00$	$r_8 = 1.00$	$r_9 = 1.00$
	$r_{10} = 1.00$	$r_{11} = 1.00$	$r_{12} = 1.00$	$r_{13} = 1.00$	

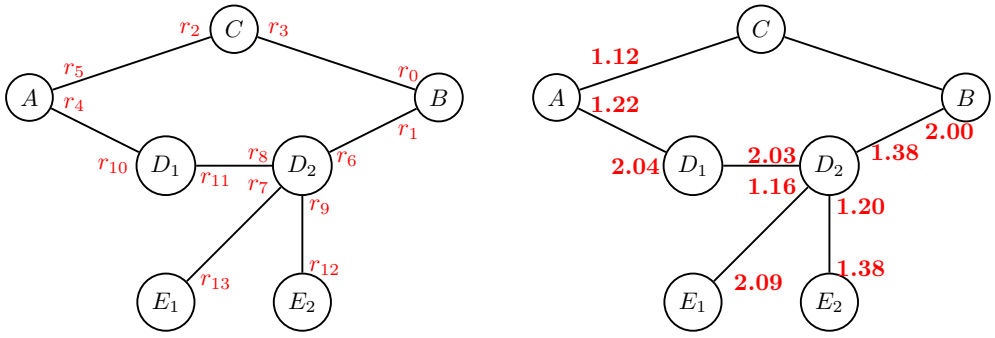
Table 6.9: List of the highest-scoring subsystems in the *Configuration Variability* metric in the Road Block scenario shortly after the first disturbance.

Analysis

When we take the sum of the scores (as provided in Table 6.10), and assign them to their respective locations in the system (shown in Figure 6.11), we see that most of the highest-scoring red lights are at lanes connected to intersection D_2. This leads to the assumption that unusual traffic patterns occur in D_2. This is, again, the closest we can get to the root cause with this approach. We have no further hint from the scores and the metrics that intersection C is the actual problem.

Step	Score sum				
258	$r_{13} = 2.09$	$r_{10} = 2.04$	$r_8 = 2.03$	$r_1 = 2.00$	$r_6 = 1.38$
	$r_{12} = 1.38$	$r_4 = 1.22$	$r_9 = 1.20$	$r_7 = 1.16$	$r_5 = 1.12$
259	$r_1 = 3.00$	$r_{13} = 2.14$	$r_8 = 2.13$	$r_7 = 2.10$	$r_{10} = 1.94$
	$r_{12} = 1.49$	$r_6 = 1.42$	$r_9 = 1.24$	$r_{11} = 1.17$	$r_3 = 0.77$

Table 6.10: List of the highest-scoring subsystems over all metrics in the Road Block scenario shortly after the first disturbance.



(a) The subsystems (b) Top ten scores

Figure 6.11: The street network for the Road Block scenario with the actually considered subsystems – the red lights for the incoming lanes at each node – and the top ten scores at $t = 258$.

6.3.4 Application Summary

We have seen that in the examined scenarios, our framework is able to provide temporal and spatial information about sources of unusual events. This information is limited to the boundaries of the system and the sources of unusual self-adaptation. A malfunction of a single subsystem or the presence of external attackers could not be discovered. Nevertheless, the framework helped with coming closer to the answers of *where* and *when* in a root cause analysis.

6.4 Other RCA Approaches

As mentioned above, the root cause analysis is a structured process. There are several methodologies and strategies that provide such a structure. The commonly used approaches include the Ishikawa Fishbone Diagram [126], the Five Why technique [204], and the Fault Tree Analysis [122]. Such strategies are only the starting point for a root cause analysis. The choice of the strategy and its actual implementation depends on the user. A comprehensive overview of tools and techniques for RCA in technical systems can be found in [213]. Building upon this toolset, many RCA implementations for technical systems have been proposed in the literature. For a comparison, we will restrict ourselves to the domains of distributed systems and self-adaptive systems.

For cluster applications, *Agarwal et al.* [2] describe an approach based on dependency graphs with a behaviour model. They measure the response times of cluster nodes and use their dependencies on each other to identify problems. *Banerjee et al.* present a framework for distributed RCA in large IP networks [9]. Their framework is another example of approaches that work with relationship graphs. The framework basically divides the

network into smaller subnetworks where centralised RCA methods are applied. A review of RCA methods based on such graphs is given by *Glymour et al.* [71].

Zhang et al. [252] present a machine-learning-based system to analyse multivariate time series for anomaly detection and diagnosis. While their starting point is comparable to the scope of this thesis, their approach assigns scores to possible anomalies based on their duration, However, they can only identify the possible time series (i.e. a sensor) that caused the anomaly, and their neural network requires training with data from the actual system. Our framework, on the other hand, requires no training and can point to spatial sources of events even in systems with moving subsystems.

Another tool for RCA based on time series is presented by *Jeyakumar et al.* [100]. Their system uses causal Bayesian networks to aid anomaly explanation in time series. For this, the user has to provide a hypothesis on causal dependencies between several time series. The Bayesian networks are then derived from this input. Furthermore, this system is only used after anomalies have been detected. The user has to provide the time series he wishes to be analysed and explained. The Bayesian networks are then again analysed with graph-based methods [71].

For cloud computing platforms, *Zhang et al.* present a framework based on Bayesian networks which is built upon several data sources like logfile and system monitoring metrics and domain knowledge. They call this type of network a knowledge-informed hierarchical Bayesian network [256].

The detection and isolation of faults in multi-agent systems have been widely regarded in research literature. Especially the domain of decentralised fault detection has been intensively investigated. An overview can be found in [177]. All approaches presented there rely on a specific property of the MAS: communication between the agents. Systems without communication or with only limited exchange of information are not covered by them.

In general, problem analysis in distributed systems either makes use of graph-theoretic approaches (causal graphs, relationship graphs, or dependency graphs) or analyses each component individually (e.g. by anomaly detection in a time series from that component). *Raj et al.* state that for adaptive distributed systems, the graph-theoretic approach is problematic because self-adaptation can cause a change in the structure of these graphs [181]. To solve this problem, they propose an RCA method that uses statecharts [88]. But this requires knowledge about system states.

In conclusion, we can say that the established RCA methods all require knowledge about the system. At this point, our measurement framework is no different. We need to configure the parameters for our framework. But this can be done with experimentation using only the output (i.e. the configurations) of the system. Other approaches require deeper insight about causalities, subsystem relationships or system states. These system-

specific RCA methods are highly optimised and work very well. Yet, our framework, when used in the context of an RCA, can assist in the analysis due to its overarching view of the system.

6.5 Comparison to a Classical Method

Now, we will see how the "Where?" can be approached with a simple data analysis method and compare the result to our framework.

In our SASO systems, the available data exists as a set of time series. When investigating the "Where?" of interesting events in such a data source, the first thing to try is anomaly detection in time series. Those time series that show an anomaly at the given time are candidates for the source of the event. To limit the number of candidates, the anomalies must be quantified, usually by assigning the distance of the anomaly to a baseline, a mean or a cluster centroid [249].

From the huge toolset of anomaly scoring methods, we choose the Histogram-based Outlier Score (HBOS) [72]. It is easy to implement, requires only basic mathematics and can be used at runtime.

After introducing HBOS in detail, we will apply it in our two RCA scenarios and compare the results with the scoring extension of our framework.

6.5.1 Histogram-based Outlier Score

HBOS builds several histograms for a given data set. If the data points are n-dimensional, then for every entry i in the data vector $v = (v_1, ..., v_n)$ a single histogram is created, where the i-th histogram contains the values v_i for all data points.

After the histograms are built, the height of the bins is normalised. The height of a bin is the number of data points in the bin. Let h be the height of the biggest bin, i.e. the number of entries in the highest-counting bin. Then, the height of bin b_i is divided by h. This way, the highest-counting bin is assigned the height 1.

The score HBOS(p) of an outlier p is calculated as

$$\text{HBOS}(p) := \sum_{i=1}^{n} log \frac{1}{hist_i(p)}, \tag{6.5}$$

where $hist_i(p)$ is the height of the bin in the histogram for the i-th dimension of the configuration vector in which the data point p lies.

6.5.2 Application in the Flocking Scenario

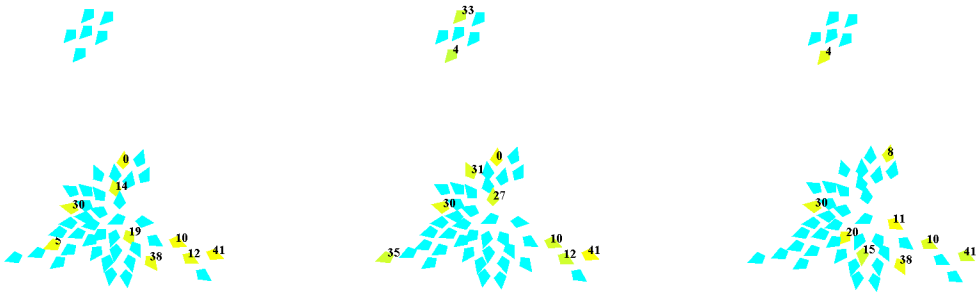
To assign a score to each bird for a given point t in time, we collect the last 20 configuration values, i.e. the bird's orientations in the time frame $[t-20, t]$. This data set of 20 points is the input of the HBOS computation, and then the score for the bird at time point t is the HBOS score for the configuration value that the bird has at that time. Note that the window size of 20 steps is the result of experimentation. The number of bins is set to 60, again as a divisor of 360, since we have angular values here.

Like above, we will limit our attention to the ten highest-scoring birds for the steps after the disturbance. Table 6.11 shows the results.

Step	**Subsystem HBOS score**				
502	$a_0 = 5.41$	$a_5 = 5.31$	$a_{10} = 5.48$	$a_{12} = 5.19$	$a_{14} = 5.10$
	$a_{19} = 5.04$	$a_{30} = 5.08$	$a_{38} = 5.19$	$a_{41} = 5.54$	$a_{47} = 5.27$
503	$a_0 = 5.27$	$a_4 = 4.24$	$a_{10} = 4.65$	$a_{12} = 4.49$	$a_{27} = 4.78$
	$a_{30} = 5.03$	$a_{31} = 4.83$	$a_{33} = 4.47$	$a_{35} = 4.49$	$a_{41} = 5.41$
504	$a_4 = 4.24$	$a_8 = 4.24$	$a_{10} = 4.10$	$a_{11} = 4.58$	$a_{15} = 3.87$
	$a_{20} = 4.43$	$a_{30} = 4.27$	$a_{38} = 4.38$	$a_{41} = 3.88$	$a_{47} = 4.02$

Table 6.11: List of the highest-scoring subsystems in the Flocking scenario shortly after the disturbance using the HBOS scoring.

This list contains some of the birds that received a top score in our framework, e.g. a_{41}, a_{19}, a_{14} and a_{12} for $t = 502$, and some birds that are not in the framework's list. Nevertheless, looking at the locations of the birds in the HBOS list, which are shown in Figure 6.12, we can deduce the spatial source by following the trajectories backwards.



(a) $t = 502$ (b) $t = 503$ (c) $t = 504$

Figure 6.12: The 10 highest-scoring birds in the Flocking scenario shortly after the disturbance using the HBOS scoring.

6.5.3 Application in the Smart Camera Scenario

In this scenario, we will again use a window size of 20 steps and a bin count of 60. These values are, again, chosen for the same reasons as above. Our point of attention is $t = 111$, one step after the beginning of the disturbance. Figure 6.13 shows the scores for each camera.

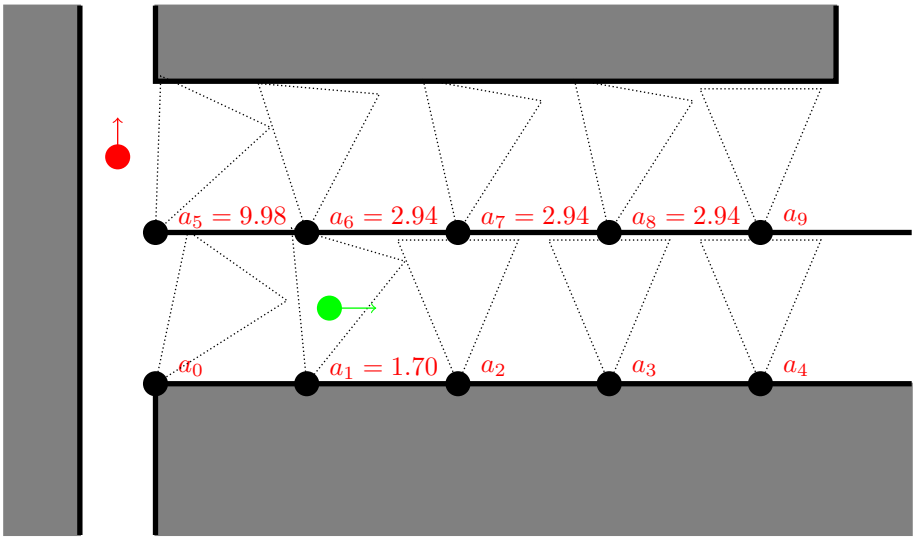


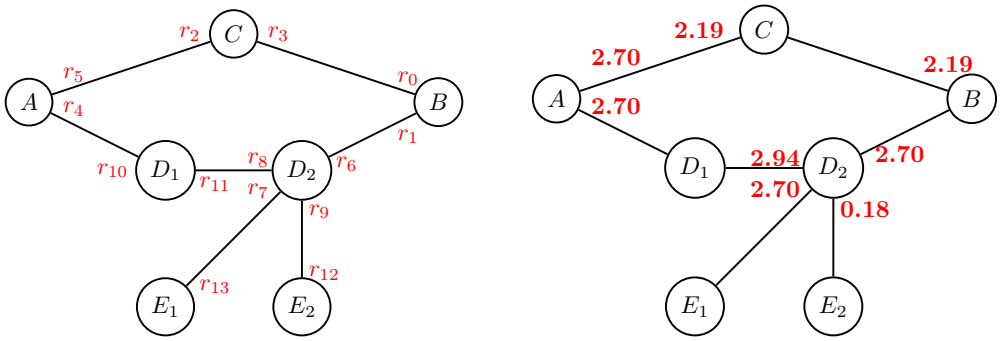
Figure 6.13: The hacked Smart Camera simulation at $t = 111$ with the resulting HBOS scores. Only the positive scores are shown.

Except for the score values, this is the same result as our framework produces in this scenario. Again, the camera a_5 is the primary candiate for the source of the disturbance.

6.5.4 Application in the Artificial Road Block Scenario

To complete the comparison, we now presented the HBOS scores in the artificial Road Block scenario. This time, the HBOS is configured with a window size of 20 steps and 40 bins. For brevity, we list the HBOS output of time step $t = 258$ and display it directly in the network picture. The result is shown in Figure 6.14.

In this case, the incoming lanes at intersection D_2 are, again, highly scored. This time, the lane from D_1 to D_2 has the second-highest score, while our framework gave it the first rank. On the other hand, the HBOS scoring assigns a score to an incoming lane at intersection C and a zero to the lanes at E_1 and E_2. Nevertheless, the HBOS score points at D_2 and A as a candidate for further investigation of the disturbance.



(a) The subsystems (b) Top ten scores

Figure 6.14: The street network for the Road Block scenario with the actually considered subsystems – the red lights for the incoming lanes at each node – and the top scores of the HBOS scoring at $t = 258$. Scores of 0 are not displayed.

6.5.5 Comparison Summary

In this section, we saw that our framework produces similar results when compared to a classical outlier scoring approach. In the Flocking scenario, the highest-scoring subsystems were not always the same in both approaches, but both approaches assigned high scores to sufficiently enough birds with a meaningful position and trajectory. In the Smart Camera scenario, our framework and the HBOS scoring came to identical results. The Artificial Road Block scenario is another example where different subsystems are in the top ten lists. However, a human expert still gets the same pointers from both scoring approaches.

6.6 RCA Summary

As we have seen, with small modifications, our framework can help in the process of a root cause analysis. Of course, this is restricted to systems where our framework is applicable and can detect unusual events sufficiently well. However, it is limited to the detection of unusual events within the technical bounds of the system. The application in the Road Block scenario shows this: although the root cause – the blockade of an entire intersection – takes place in one location in the system, the unusual event, in the view of self-adaptation, occurs somewhere else. Therefore, the framework cannot replace established RCA approaches which use specific knowledge about the system. Furthermore, we have compared our scoring approach to a classical idea. While in the classical approach one score is assigned to each time series, our framework assigns several distinct scores, one for each metric. Nevertheless, the results are comparable. This shows that the scoring in our metrics can be used in the same way as other methods and that

our metrics and existing scorings can complement each other for root cause analysis applications.

Chapter 7

Conclusion

You have a lot of measurements.
Quite a few variables.

Randall Munroe[1]

Let us now conclude this thesis with a summary and an outlook on future work.

7.1 Summary

This thesis presented a framework of several metrics that helps in the assessment of the self-adaptation behaviour of SASO systems. We started with the core idea that unusual self-adaptation is reflected in unusual changes in the configurations of the subsystem. We described how the framework applies existing and new metrics to the configurations and their time series to create a single time series of every metric. In addition, the framework incorporates a metric that analyses the communication messages in the SASO system.

Then, we evaluated the framework in eight scenarios from different domains that were chosen to cover a wide range of system properties, such as openness and closeness to real-world systems. We introduced the scenarios in detail and showed how a disturbance in the systems was created to lead the systems to unusual self-adaptation. For each scenario, the results of every metric and the influence of their parameters were presented and discussed. Afterwards, we summarised the evaluation and compared the results across the several scenarios.

We then presented a possibility of extending the framework with a scoring feature and showed how this extended framework can be applied as a tool in a root cause analysis. We evaluated the augmented framework in three of the previously introduced scenarios with the goal of identifying a potential spatial source for the detected unusual self-adaptation event.

The contents of this thesis provide the following contributions to research:

[1] The *xkcd.com* webcomic [271].

- **Metrics for self-adaptation:** We created new metrics that are able to measure self-adaptation with a unique focus on the configurations. These metrics are designed to assess the self-adaptation at the system-wide level. Furthermore, we created a framework to combine and simultaneously apply the metrics. Due to their reduced requirement for system knowledge, the metrics in the framework can be applied to a broad range of systems from various domains.
- **Behavioural analysis of SASO systems:** The framework can aid in a behavioural analysis by detecting changes in a system that might lead to a new observable behaviour. An early indication for possible regional or global behaviour changes resulting from unusual self-adaptation can help to anticipate possible problems.
- **Root cause analysis:** With small extensions, the framework can help in root cause analysis by supporting system-specific methods with additional insights into possible sources of problems.

7.2 Room for More

This thesis provided the formal foundations for the framework. Although we evaluated it in scenarios with real-world data, there are still some open questions that have to be considered before the framework can be implemented for an actual real-world SASO system. The questions that should be approached in future work are:

- For the actual system, we have to decide what we consider as the configuration that we want to analyse. In simple systems, such as the birds in the Flocking scenario, there is not much room for a choice. However, when we consider SASO systems where the productive system decides its actions with a ruleset and where the control mechanism can change this ruleset, it is possible to use a suitable numerical representation of the rulesets as the configuration instead of the output of the current ruleset.
- How to deal with possible problems with the acquisition of the configurations of the subsystems? There can be delays in the response of a subsystem, general asynchronisms in the communication, and data can be corrupted or missing.
- We evaluated the framework in simulations with discrete time steps. How can the framework handle systems with continuous timings?
- The framework collects the configurations from all subsystems to create a global analysis. In very large systems, problems might be limited to a specific region. Thus, it should be investigated how the framework can be implemented on a regional level. Furthermore, in very large systems with possibly millions of subsystems, the computational performance needs to be considered.

- In our scenarios, we have seen that the detection performance of the framework depends on the system. The evaluation should be extended to more and different systems. A classification of system properties that reduce or improve the detection performance would be beneficial for application attempts.
- During the evaluation and the application in the RCA scenarios, we used a peak detection algorithm, the comparison with expected time series and the human eye to identify anomalies in the output of the framework. It should be investigated which existing methods for time series analysis can further improve the event detection performance.
- The detection performance in the scenarios that we have classified as open systems is relatively poor with the given metrics. Additional metrics should be researched to improve the detection quality for such systems.
- It is possible that SASO systems will encounter other SASO systems in their environment. Their actions will most likely lead to a mutual influence on their behaviour. Here, the question is how our framework can support existing detection algorithms for mutual influence [196] or the learning algorithms under mutual influence [194, 195].
- How can existing anomaly detection algorithms and novelty detection methods which do not meet the requirements of the framework be adapted to be included? A starting point in this direction can be heuristics for the online applicability [4, 82] of such methods.

References

[1] Omid R. Abbasi, Ali A. Alesheikh, Mina Karimi, and Sina Abolhoseini. Location Recommendation in Geo-Social Networks: A Human-Centric Agent-Based Approach. In *Adjunct Proceedings of the 14th International Conference on Location Based Services*, pages 189–193. ETH Zurich, 2018.

[2] Manoj K. Agarwal, Karen Appleby, Manish Gupta, Gautam Kar, Anindya Neogi, and Anca Sailer. Problem determination using dependency graphs and run-time behavior models. In *Utility Computing: 15th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2004, Davis, CA, USA, November 15-17, 2004. Proceedings 15*, pages 171–182. Springer, 2004.

[3] Nathan Aky, Denis Payet, Sylvain Giroux, and Rémy Courdier. Ubiquitous Computing and Multi-agent Systems: Clarification of the Lexicon. In *PRIMA 2019: Principles and Practice of Multi-Agent Systems: 22nd International Conference, Turin, Italy, October 28–31, 2019, Proceedings 22*, pages 602–609. Springer, 2019.

[4] Ghassan Al-Falouji, Christian Gruhl, Torben Neumann, and Sven Tomforde. A Heuristic for an Online Applicability of Anomaly Detection Techniques. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 107–112. IEEE, 2022.

[5] Ghassan Al-Falouji, Lukas Haschke, Dirk Nowotka, and Sven Tomforde. Self-Explanation as a Basis for Self-Integration – The Autonomous Passenger Ferry Scenario. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, accepted for publication.

[6] Ricardo Alvira. *The definition of Complex and Complexity*, page 12. CreateSpace Independent Publishing Platform, 09 2014.

[7] William Ross Ashby. *An introduction to cybernetics*. Chapman & Hall Ltd., 1957.

[8] William Ross Ashby. Principles of the self-organizing system. In *Systems research for behavioral science*, pages 108–118. Routledge, 2017.

[9] Dipyaman Banerjee, Venkateswara Madduri, and Mudhakar Srivatsa. A framework for distributed monitoring and root cause analysis for large ip networks. In *2009 28th IEEE International Symposium on Reliable Distributed Systems*, pages 246–255. IEEE, 2009.

[10] Wolfgang Banzhaf. Self-organizing Systems. *Encyclopedia of complexity and systems science*, 14:589, 2009.

[11] Jaime Barceló and Jordi Casas. Dynamic network simulation with AIMSUN. In *Simulation approaches in transportation analysis: Recent advances and challenges*, pages 57–98. Springer, 2005.

[12] Birger Becker, Florian Allerding, Ulrich Reiner, Mattias Kahl, Urban Richter, Daniel Pathmaperuma, Hartmut Schmeck, and Thomas Leibfried. Decentralized energy-management to control smart-home architectures. In *Architecture of Computing Systems-ARCS 2010: 23rd International Conference, Hannover, Germany, February 22-25, 2010. Proceedings 23*, pages 150–161. Springer, 2010.

[13] Matthias Becker, Markus Luckey, and Steffen Becker. Performance analysis of self-adaptive systems for requirements validation at design-time. In *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, pages 43–52, 2013.

[14] Jenay Beer, Arthur Fisk, and Wendy Rogers. Toward a Framework for Levels of Robot Autonomy in Human-Robot Interaction. *Journal of Human-Robot Interaction*, 3:74, 06 2014.

[15] Jan Beirlant, Edward J Dudewicz, László Györfi, and Edward Van der Meulen. Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39, 1997.

[16] Kirstie Bellman, Jean Botev, Ada Diaconescu, Lukas Esterle, Christian Gruhl, Christopher Landauer, Peter Lewis, Phyllis Nelson, Evangelos Pournaras, Anthony Stein, and Sven Tomforde. Self-improving system integration: Mastering continuous change. *Future Gener. Comput. Syst.*, 117:29–46, 2021.

[17] Kirstie Bellman, Sven Tomforde, and Rolf P. Würtz. Interwoven Systems: Self-Improving Systems Integration. In *Eighth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2014, London, United Kingdom, September 8-12, 2014*, pages 123–127. IEEE Computer Society, 2014.

[18] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[19] Nelly Bencomo, Kris Welsh, Pete Sawyer, and Jon Whittle. Self-explanation in adaptive systems. In *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, pages 157–166. IEEE, 2012.

[20] Andrew Berns and Sukumar Ghosh. Dissecting self-* properties. In *2009 Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 10–19. IEEE, 2009.

[21] Lachlan Birdsey, Claudia Szabo, and Katrina Falkner. Identifying self-organization and adaptability in complex adaptive systems. In *2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 131–140. IEEE, 2017.

[22] Christopher Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2nd edition, 2011.

[23] Hartmut Bossel. Goal Functions and Orientors. *Encyclopedia of Ecology*, pages 1746–1754, 12 2008.

[24] Michael E. Bratman. *Intention, Plans, and Practical Reason*. Cambridge University Press, March 1999.

[25] Tom Broekel. Measuring technological complexity - Current approaches and a new measure of structural complexity. *arXiv preprint arXiv:1708.07357*, 2017.

[26] Daniel Brown and Michael Goodrich. Limited bandwidth recognition of collective behaviors in bio-inspired swarms. *13th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2014*, 1:405–412, 01 2014.

[27] Emre Cakar, Jörg Hähner, and Christian Müller-Schloer. Investigation of generic observer/controller architectures in a traffic scenario. *INFORMATIK 2008. Beherrschbare Systeme-dank Informatik. Band 2*, 2008.

[28] Javier Cámara, Pedro Correia, Rogério de Lemos, and Marco Vieira. Empirical resilience evaluation of an architecture-based self-adaptive software system. In *Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures*, pages 63–72, 2014.

[29] George Candea, James Cutler, and Armando Fox. Improving availability with recursive microreboots: A soft-state system case study. *Performance Evaluation*, 56:213–248, 10 2003.

[30] Rafael Cardoso and Angelo Ferrando. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers*, 10:16, 01 2021.

[31] Valentina Castiglioni, Michele Loreti, and Simone Tini. Measuring adaptability and reliability of large scale systems. In *Leveraging Applications of Formal Methods, Verification and Validation: Engineering Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part II 9*, pages 380–396. Springer, 2020.

[32] Gianpiero Cattaneo, Alberto Dennunzio, and Fabio Farina. A full cellular automaton to simulate predator-prey systems. In *International Conference on Cellular Automata*, pages 446–451. Springer, 2006.

[33] Renyue Cen. Temporal Self-Organization in Galaxy Formation. *The Astrophysical Journal Letters*, 785, 03 2014.

[34] Wai Kin Victor Chan. Interaction metric of emergent behaviors in agent-based simulation. In *Proceedings of the 2011 Winter Simulation Conference (WSC)*, pages 357–368. IEEE, 2011.

[35] David M Chess, Alla Segal, Ian Whalley, and Steve R White. Unity: Experiences with a prototype autonomic computing system. In *International Conference on Autonomic Computing, 2004. Proceedings.*, pages 140–147. IEEE, 2004.

[36] Paul Cilliers. *Complexity and postmodernism: Understanding complex systems*. routledge, 2002.

[37] Michel Cotsaftis. What Makes a System Complex? - An Approach to Self Organization and Emergence. *From System Complexity to Emergent Properties*, pages 49–99, 2009.

[38] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2012.

[39] Mark Cross and Pierre Hohenberg. Pattern Formation Outside of Equilibrium. *Rev. of Mod. Phys.*, 65:851–1112, 07 1993.

[40] James Crutchfield and Karl Young. Inferring statistical complexity. *Physical review letters*, 63:105–108, 08 1989.

[41] Maike Bezerra da Silva, Carla Bezerra, Emanuel Coutinho, and Paulo Henrique Maia. A Catalog of Performance Measures for Self-Adaptive Systems. In *Proceedings of the XX Brazilian Symposium on Software Quality*, pages 1–10, 2021.

[42] Anwesha Das, Daniel Ratner, and Alex Aiken. Performance Variability and Causality in Complex Systems. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 19–24. IEEE, 2022.

[43] Jennifer Deaton and Randal Allen. Development and Application of System Complexity Measures for Use in Modeling and Simulation. In *Proceedings of the Conference on Summer Computer Simulation*, SummerSim '15, page 1–6, San Diego, CA, USA, 2015. Society for Computer Simulation International.

[44] Gerard Debreu. Representation of a Preference Ordering by a Numerical Function. *Decision Processes*, 01 1954.

[45] Pooja Dehraj and Arun Sharma. A review on architecture and models for autonomic software systems. *The Journal of Supercomputing*, 77, 01 2021.

[46] Ute Deichmann. The idea of self-organisation in biology and its critics. *Open Access Government*, 38:306–307, 04 2023.

[47] Marianne Delorme. An introduction to cellular automata: some basic definitions and concepts. In *Cellular automata: a parallel model*, pages 5–49. Springer, 1999.

[48] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Karageorgos Anthony. Self-Organisation in MAS. *The Knowledge Engineering Review*, 20:165–189, 08 2005.

[49] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes, and Karageorgos Anthony. Self-Organisation and Emergence in MAS: An overview. *Informatica*, 30, 01 2006.

[50] Andrew Dillon and Michael Morris. User Acceptance of Information Technology: Theories and Models. *Annual Review of Information Science and Technology*, 31, 01 1996.

[51] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Einführung in die mathematische Logik*, volume 3. Springer, 1992.

[52] Benedikt Eberhardinger, Gerrit Anders, Hella Seebach, Florian Siefert, and Wolfgang Reif. A research overview and evaluation of performance metrics for self-organization algorithms. In *2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 122–127. IEEE, 2015.

[53] Benedikt Eberhardinger, Hella Ponsar, Dominik Klumpp, and Wolfgang Reif. Measuring and Evaluating the Performance of Self-Organization Mechanisms Within Collective Adaptive Systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems*, pages 202–220, Cham, 2018. Springer International Publishing.

[54] Wilfried Elmenreich and Hermann de Meer. Self-organizing networked systems for technical applications: A discussion on open issues. In *International Workshop on Self-Organizing Systems*, pages 1–9. Springer, 2008.

[55] Wilfried Elmenreich, Raissa D'Souza, Christian Bettstetter, and Hermann de Meer. A survey of models and design methods for self-organizing networked systems. In *International Workshop on Self-Organizing Systems*, pages 37–49. Springer, 2009.

[56] Dominik Fisch, Martin Janicke, Bernhard Sick, and Christian Muller-Schloer. Quantitative Emergence–A Refined Approach Based on Divergence Measures. In *2010 Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 94–103. IEEE, 2010.

[57] Christopher Flathmann, Nathan McNeese, and Lorenzo Barberis Canonico. Using Human-Agent Teams to Purposefully Design Multi-Agent Systems. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 63:1425–1429, 11 2019.

[58] Otto Forster. *Analysis 2: Differentialrechnung im IRn, gewöhnliche Differentialgleichungen*. Analysis / Otto Forster. Vieweg+Teubner Verlag, 2008.

[59] H.I. Freedman and Paul Waltman. Persistence in Models of Three Interacting Predator-Prey Populations. *Mathematical Biosciences*, 68:213–231, 04 1984.

[60] Christian Fuchs. Concepts of Social Self-Organisation. *SSRN eLibrary*, 08 2002.

[61] Mathematical Games. The fantastic combinations of John Conway's new solitaire game "life" by Martin Gardner. *Scientific American*, 223:120–123, 1970.

[62] Zhiqiang Gao. Engineering cybernetics: 60 years in the making. *Control Theory and Technology*, 12:97–109, 05 2014.

[63] Murray Gell-Mann. *Complex adaptive systems*. Addison-Wesley, 1994.

[64] Ilias Gerostathopoulos, Claudia Raibulet, and Elvin Alberts. Assessing self-adaptation strategies using cost-benefit analysis. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 92–95. IEEE, 2022.

[65] Ilias Gerostathopoulos, Thomas Vogel, Danny Weyns, and Patricia Lago. How do we evaluate self-adaptive software systems?: A ten-year perspective of SEAMS. In *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 59–70. IEEE, 2021.

[66] Carlos Gershenson and Nelson Fernández. Complexity and Information: Measuring Emergence, Self-organization, and Homeostasis at Multiple Scales. *Complexity*, 18, 11 2012.

[67] Marius Ghergu and Vicentiu D. Radulescu. *Reaction-Diffusion Systems Arising in Chemistry*, pages 287–335. Springer, 2012.

[68] Ciprian Doru Giurcăneanu, Panu Luosto, and Petri Kontkanen. On the performance of histogram-based entropy estimators. In *2012 IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6. IEEE, 2012.

[69] Torkel Glad and Lennart Ljung. *Control theory*. CRC press, 2018.

[70] Robert L. Glass. *Facts and Fallacies of Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[71] Clark Glymour, Kun Zhang, and Peter Spirtes. Review of causal discovery methods based on graphical models. *Frontiers in genetics*, 10:524, 2019.

[72] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

[73] Martin Goller, Ingo Thomsen, Ghassan Al-Falouji, and Sven Tomforde. Abnormal Behaviour Detection of Self-Adaptive Agents in Traffic Environments. In *2023 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, accepted for publication.

[74] Martin Goller and Sven Tomforde. Towards a continuous assessment of stability in (self-) adaptation behaviour. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 154–159. IEEE, 2020.

[75] Martin Goller and Sven Tomforde. Assessment of Configuration Stability and Variability in Collections of Self-Adaptive Systems. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 125–130, 2021.

[76] Martin Goller and Sven Tomforde. Beyond Homeostasis: A Novel Approach for Assessing the Stability and Coherence of Self-Adaptive Systems. In *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, pages 10–17. IEEE, 2021.

[77] Martin Goller and Sven Tomforde. On the Stability of (Self-)Adaptive Behaviour in Continuously Changing Environments: A Quantification Approach. *Array*, 11:100069, 05 2021.

[78] Martin Goller and Sven Tomforde. Identifying Adaptation Changes in Collections of Self-Adaptive Systems. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 101–106. IEEE, 2022.

[79] Martin Goller and Sven Tomforde. Runtime Assessment of the Parameter Utilisation in Adaptive Systems. In *2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pages 200–205. IEEE, 2022.

[80] Charles Gouin-Vallerand, Bessam Abdulrazak, Sylvain Giroux, and Mounir Mokhtari. Toward Autonomic Pervasive Computing. In *Proceedings of the 10th International Conference on Information Integration and Web-Based Applications & Services*, iiWAS '08, page 673–676, New York, NY, USA, 2008. Association for Computing Machinery.

[81] Artur Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects*. Studies in Big Data. Springer International Publishing, 2019.

[82] Christian Gruhl and Sven Tomforde. OHODIN–online anomaly detection for data streams. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 193–197. IEEE, 2021.

[83] Raphael Götz, Christian Krupitzer, and Samuel Kounev. Survey on the Evaluation of Self-Aware Computing Systems. Master's thesis, Julius-Maximilians-Universität Würzburg, 2020.

[84] Marwa Hachicha, Riadh Ben Halima, and Ahmed Hadj Kacem. Formal Verification approaches of Self-adaptive Systems: A Survey. *Procedia Computer Science*, 159:1853–1862, 2019. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[85] Julianne D. Halley and David A. Winkler. Consistent concepts of self-organization and self-assembly. *Complexity*, 14(2):10–17, 2008.

[86] Greg Hamerly and Charles Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16, 2003.

[87] Uwe Hansmann, Lothar Merk, Martin S Nicklous, and Thomas Stober. *Pervasive Computing: The Mobile World*. Springer Professional Computing. Springer Berlin Heidelberg, 2003.

[88] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[89] Klaus Herrmann, Gero Mühl, and Kurt Geihs. Self-Management: The Solution to Complexity or Just Another Problem? *Distributed Systems Online, IEEE*, 6, 02 2005.

[90] Francis Heylighen. The science of self-organization and adaptivity. *The encyclopedia of life support systems*, 5(3):253–280, 2001.

[91] Michael Hinchey, James Rash, Walter Truszkowski, Christopher Rouff, and Roy Sterritt. You can't get there from here! Problems and some potential solutions in developing new classes of complex systems. *Journal of Integrated Design & Process Science*, 9:1–16, 01 2005.

[92] Michael Hinchey and Roy Sterritt. Self-Managing Software. *Computer*, 39:107 – 109, 03 2006.

[93] Richard Holzer and Hermann de Meer. Methods for approximations of quantitative measures in self-organizing systems. In *Self-Organizing Systems: 5th International Workshop, IWSOS 2011, Karlsruhe, Germany, February 23-24. 2011. Proceedings 5*, pages 1–15. Springer, 2011.

[94] Paul Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. *Computing Systems*, 2007, 10 2001.

[95] Kasper Hornbæk and Morten Hertzum. Technology acceptance and user experience: A review of the experiential component in HCI. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 24(5):1–30, 2017.

[96] Philippe Huneman. Emergence and Adaptation. *Minds and Machines*, 18:493–520, 12 2008.

[97] Christian Janiesch, Marcus Fischer, Axel Winkelmann, and Valentin Nentwich. Specifying autonomy in the Internet of Things: the autonomy model and notation. *Information Systems and e-Business Management*, 03 2019.

[98] Nicholas R Jennings. Building complex software systems: The case for an agent-based approach. *Communications of the ACM*, 44:35–41, 2001.

[99] Nicholas R. Jennings, Jose Manuel Corera, and Inaki Laresgoiti. Developing Industrial Multi-Agent Systems. In *ICMAS*, pages 423–430, 1995.

[100] Vimalkumar Jeyakumar, Omid Madani, Ali Parandeh, Ashutosh Kulshreshtha, Weifei Zeng, and Navindra Yadav. ExplainIt!–A declarative root-cause analysis engine for time series data. In *Proceedings of the 2019 International Conference on Management of Data*, pages 333–348, 2019.

[101] Neil Johnson. *Simply Complexity: A Clear Guide to Complexity Theory*. Oneworld, 2009.

[102] Elsy Kaddoum, Claudia Raibulet, Jean-Pierre Georgé, Gauthier Picard, and Marie-Pierre Gleizes. Criteria for the evaluation of self-* systems. *Proceedings - International Conference on Software Engineering*, pages 29–38, 05 2010.

[103] Jan Kantert, Sven Tomforde, and Christian Mueller-Schloer. Measuring self-organisation in distributed systems by external observation. In *ARCS 2015-The 28th International Conference on Architecture of Computing Systems. Proceedings*, pages 1–8. VDE, 2015.

[104] Edward Kaplan and Hsue-Shen Tsien. Engineering Cybernetics. *The American Mathematical Monthly*, 62:667, 11 1955.

[105] Jeffrey Kephart and David M. Chess. The Vision Of Autonomic Computing. *Computer*, 36:41 – 50, 02 2003.

[106] David King and Gilbert Peterson. A macro-level order metric for self-organizing adaptive systems. In *2018 IEEE 12th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 60–69. IEEE, 2018.

[107] Tetsuo Kinoshita. Basic characteristics of a macroscopic measure for detecting abnormal changes in a multiagent system. *Sensors*, 15(4):9112–9135, 2015.

[108] Verena Klös, Thomas Göthel, and Sabine Glesner. Formal models for analysing dynamic adaptation behaviour in real-time systems. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 106–111. IEEE, 2016.

[109] David B Knoester, Heather J Goldsby, and Christoph Adami. Leveraging evolutionary search to discover self-adaptive and self-organizing cellular automata. *arXiv preprint arXiv:1405.4322*, 2014.

[110] Khairy Kobbacy, Prabhakar Murthy, et al. *Complex system maintenance handbook*. Springer, 2008.

[111] Jana Koehler, Chris Giblin, Dieter Gantenbein, and Rainer Hauser. On autonomic computing architectures. *IBM Research, Zurich Research Laboratory*, 2003.

[112] Andrei N. Kolmogorov. On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376, 1963.

[113] Milan Kotva. Medical cybernetics. *Ceskoslovenské zdravotnictví*, 36:193–202, 06 1988.

[114] Samuel Kounev, Jeffrey Kephart, Aleksandar Milenkoski, and Xiaoyun Zhu. *Self-Aware Computing Systems*. Springer, 02 2017.

[115] Christian Krupitzer, Martin Pfannemüller, Vincent Voss, and Christian Becker. Comparison of Approaches for developing Self-adaptive Systems. 2018.

[116] Christian Krupitzer, Felix Roth, Martin Pfannemüller, and Christian Becker. Comparison of approaches for self-improvement in self-adaptive systems. In *2016 IEEE international conference on autonomic computing (ICAC)*, pages 308–314. IEEE, 2016.

[117] Christian Krupitzer, Felix Roth, Sebastian VanSyckel, Gregor Schiele, and Christian Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17, 10 2014.

[118] Philippe Lalanda, Julie A. McCann, and Ada Diaconescu. *Autonomic computing: principles, design and implementation*. Springer Science & Business Media, 2013.

[119] J. Stephen Lansing. Complex Adaptive Systems. *Annual Review of Anthropology*, 32(1):183–204, 2003.

[120] Alan Lee. Circular data. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):477–486, 2010.

[121] Jin Gyu Lee and Hyungbo Shim. Design of Heterogeneous Multi-agent System Multi-agent system for Distributed Computation. *Trends in Nonlinear and Adaptive Control: A Tribute to Laurent Praly for his 65th Birthday*, pages 83–108, 2021.

[122] Wen-Shing Lee, Doris Grosh, Frank Tillman, and Chang Lie. Fault tree analysis, methods, and applications - A review. *IEEE transactions on reliability*, 34(3):194–203, 1985.

[123] Biagio Lenzitti, Domenico Tegolo, and Cesare Valenti. Prey-predator strategies in a multiagent system. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, pages 184–189. IEEE, 2005.

[124] Peter Lewis, Arjun Chandra, Shaun Parsons, Edward Robinson, Kyrre Glette, Rami Bahsoon, and Jim Torresen. A Survey of Self-Awareness and Its Application in Computing Systems. *Proceedings - 2011 5th IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2011*, 10 2011.

[125] Peter Lewis, Arjun Chandra, Shaun Parsons, Edward Robinson, Kyrre Glette, Rami Bahsoon, Jim Torresen, and Xin Yao. A survey of self-awareness and its application in computing systems. In *2011 Fifth IEEE conference on self-adaptive and self-organizing systems workshops*, pages 102–107. IEEE, 2011.

[126] Luca Liliana. A new model of Ishikawa diagram for quality assessment. *IOP Conference Series: Materials Science and Engineering*, 161(1):012099, nov 2016.

[127] Zhigui Lin, Michael Pedersen, and Lai Zhang. A predator–prey system with stage-structure for predator and nonlocal delay. *Nonlinear Analysis: Theory, Methods & Applications*, pages 2019–2030, 01 2010.

[128] Hua Liu and Manish Parashar. Accord: A programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36:341–352, 06 2006.

[129] Xinxin Liu and Siyu Liu. Dynamics of a predator–prey system with inducible defense and disease in the prey. *Nonlinear Analysis: Real World Applications*, 71:103802, 06 2023.

[130] Anna Lokrantz, Emil Gustavsson, and Mats Jirstrand. Root cause analysis of failures and quality deviations in manufacturing using machine learning. *Procedia Cirp*, 72:1057–1062, 2018.

[131] Michael Luck, Peter McBurney, and Chris Preist. *Agent technology: enabling next generation computing (a roadmap for agent based computing)*. AgentLink, 2003.

[132] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

[133] Qiuping Ma, Hongyan Li, and Anders Thorstenson. A big data-driven root cause analysis system: Application of Machine Learning in quality problem solving. *Computers & Industrial Engineering*, 160:107580, 2021.

[134] Frank Macías-Escrivá, Rodolfo Elias Haber Guerra, Raúl del Toro, and Vicente Hernández. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40:7267–7279, 12 2013.

[135] Claudio Mandrioli and Martina Maggio. Testing Self-Adaptive Software With Probabilistic Guarantees on Performance Metrics: Extended and Comparative Results. *IEEE Transactions on Software Engineering*, PP:1–1, 07 2021.

[136] Robert Marks. Analysis and synthesis: Multi-agent systems in the social sciences. *The Knowledge Engineering Review*, 27, 06 2012.

[137] Dhendra Marutho, Sunarna Hendra Handaka, Ekaprana Wijaya, and Muljono. The Determination of Cluster Number at k-Mean Using Elbow Method and Purity Evaluation on Headline News. In *2018 International Seminar on Application for Technology of Information and Communication*, pages 533–538, 2018.

[138] James Clerk Maxwell. I. On governors. *Proceedings of the Royal Society of London*, pages 270 – 283, 1868.

[139] William McElroy. Cybernetics and Biology. *Journal of Cybernetics*, 1971:1–4, 04 2008.

[140] Don McGreal and Ralph Jocham. *The Professional Product Owner: Leveraging Scrum as a Competitive Advantage*. Professional scrum series. Addison-Wesley, 2018.

[141] John Meyer. On Evaluating the Performability of Degradable Computing Systems. *Computers, IEEE Transactions on*, C-29:720–731, 09 1980.

[142] Nadine E Miner, John H Gauthier, Michael L Wilson, Hai D Le, Gio K Kao, Darryl J Melander, and Dennis E Longsine. Measuring the adaptability of systems of systems. *Military Operations Research*, 20(3):25–37, 2015.

[143] Moez Mnif and Christian Müller-Schloer. Quantitative emergence. In *Organic Computing — A Paradigm Shift for Complex Systems*, pages 39–52. Springer, 2011.

[144] Richard Mogford, J. Guttman, S. Morrow, and Parimal Kopardekar. The Complexity Construct in Air Traffic Control: A Review and Synthesis of the Literature. *Storming Media*, page 32, 07 1995.

[145] Gordon. E. Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):114 – 117, 1965.

[146] Christian Müller-Schloer. Organic computing: on the feasibility of controlled emergence. In *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 2–5, 2004.

[147] Christian Müller-Schloer and Sven Tomforde. *Organic Computing - Technical Systems for Survival in the Real World*. Birkhäuser, 2017.

[148] David J. Musliner, Scott E. Friedman, Tom Marble, Jeffrey M. Rye, Michael W. Boldt, and Michael Pelican. Self-adaptation metrics for active cybersecurity. In *2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, pages 53–58. IEEE, 2013.

[149] Facundo Mémoli. Gromov–Wasserstein Distances and the Metric Approach to Object Matching. *Foundations of Computational Mathematics*, 11:417–487, 08 2011.

[150] Dietmar Möller and Roland Haas. *The Connected Car*, pages 171–264. Springer, 04 2019.

[151] Manuel Müller, Timo Müller, Behrang Ashtari, Philipp Marks, Nasser Jazdi, and Michael Weyrich. Industrial autonomous systems: a survey on definitions, characteristics and abilities. *at - Automatisierungstechnik*, 69:3–13, 01 2021.

[152] Kai Nagel and Michael Schreckenberg. A cellular automaton model for freeway traffic. *Journal de Physique I*, 2:2221, 12 1992.

[153] Hiroyuki Nakagawa, Akihiko Ohsuga, and Shinichi Honiden. Constructing Self-Adaptive Systems Using a KAOS Model. In *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 132–137, 2008.

[154] Moeen Ali Naqvi, Sehrish Malik, Merve Astekin, and Leon Moonen. On Evaluating Self-Adaptive and Self-Healing Systems using Chaos Engineering. In *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pages 1–10. IEEE, 2022.

[155] Waldemar Nawrocki. Physical limits for scaling of integrated circuits. *Journal of Physics: Conference Series*, 248, 11 2010.

[156] Eva Nieuwdorp. The pervasive discourse. *Computers in Entertainment (CIE)*, 5:13, 04 2007.

[157] Arisa Okumara. Crystal self-organization. *Young Scientists Journal*, 6:46, 01 2013.

[158] Eugenio Oliveira and Ana Rocha. *Agents advanced features for negotiation in electronic commerce and virtual organisations formation process*, pages 78–97. Springer, 2001.

[159] Eamonn O'Toole, Vivek Nallur, and Siobhán Clarke. Decentralised Detection of Emergence in Complex Adaptive Systems. *ACM Transactions on Autonomous and Adaptive Systems*, 12:1–31, 04 2017.

[160] Mathias Pacher. Two-Level Extensions of an Artifical Hormone System. *Concurrency and Computation: Practice and Experience*, 28, 03 2015.

[161] Constantin-Valentin Pal, Florin Leon, Marcin Paprzycki, and Maria Ganzha. A Review of Platforms for the Development of Agent Systems. *CoRR*, abs/2007.08961, 2020.

[162] Girish Palshikar et al. Simple algorithms for peak detection in time-series. In *Proc. 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence*, volume 122, 2009.

[163] Liviu Panait and Sean Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 11 2005.

[164] Guiseppe Pandolfo, Antonio D'Ambrosio, and Giovanni Porzio. A note on depth-based classification of circular data. *Electronic Journal of Applied Statistical Analysis*, 11:447–462, 01 2018.

[165] Murari Parashar, Hua-Fen Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and Salim Hariri. AutoMate: Enabling Autonomic Applications on the Grid. *Cluster Computing*, 9:161–174, 01 2006.

[166] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *Proceedings of the fifth international conference on Autonomous agents*, pages 124–130, 2001.

[167] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

[168] Mohammad Farhad Peerally, Susan Carr, Justin Waring, and Mary Dixon-Woods. The problem with root cause analysis. *BMJ quality & safety*, 26(5):417–422, 2017.

[169] Dan Pelleg and Andrew Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. *Machine Learning, p*, 01 2002.

[170] Mikel D. Petty. Modeling and validation challenges for complex systems. *Spring Simulation Interoperability Workshop 2012, 2012 Spring SIW*, pages 178–187, 01 2012.

[171] Duc Truong Pham, Stefan S Dimov, and Chi D. Nguyen. Selection of K in K-means clustering. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 219(1):103–119, 2005.

[172] Jason Porter, Daniel A Menascé, Hassan Gomaa, and Emad Albassam. TESS: Automated performance evaluation of self-healing and self-adaptive distributed software systems. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 40–47, 2018.

[173] A. K. Prakash and Hrishikesh Venkataraman. Technological Challenges in Autonomous Driving. In *Proceedings of the 1st International Workshop on Communication and Computing in Connected Vehicles and Platooning*, pages 33–33, 2018.

[174] Mikhail Prokopenko, Fabio Boschetti, and Alex Ryan. An Information-Theoretic Primer on Complexity, Self-Organization, and Emergence. *Complexity*, 15:11–28, 09 2009.

[175] Mikhail Prokopenko and Carlos Gershenson. Entropy Methods in Guided Self-Organisation. *Entropy*, 16:5232–5241, 10 2014.

[176] Holger Prothmann, Sven Tomforde, Jürgen Branke, Jörg Hähner, Christian Müller-Schloer, and Hartmut Schmeck. *Organic traffic control*. Springer, 2011.

[177] Yue Quan, Wen Chen, Zhihai Wu, and Li Peng. Distributed fault detection for second-order delayed multi-agent systems with adversaries. *IEEE Access*, 5:16478–16483, 2017.

[178] Edward Raff. JSAT: Java Statistical Analysis Tool, a Library for Machine Learning. *Journal of Machine Learning Research*, 18(23):1–5, 2017.

[179] Claudia Raibulet, Arcelli Fontana, Rafael Capilla, and Carlos Carrillo. An overview on quality evaluation of self-adaptive systems. *Managing trade-offs in adaptable software architectures*, pages 325–352, 2017.

[180] Claudia Raibulet and Laura Masciadri. Metrics for the evaluation of adaptivity aspects in software systems. *International Journal on Advances in Software Volume 3, Number 1 & 2, 2010*, 2010.

[181] Amit Raj, Stephen Barrett, and Siobhan Clarke. Run-time root cause analysis in adaptive distributed systems. In *On the Move to Meaningful Internet Systems: OTM 2013 Workshops: Confederated International Workshops: OTM Academy, OTM Industry Case Studies Program, ACM, EI2N, ISDE, META4eS, ORM, SeDeS, SINCOM, SMS, and SOMOCO 2013, Graz, Austria, September 9-13, 2013, Proceedings*, pages 292–301. Springer, 2013.

[182] Anand Rao and Michael Georgeff. BDI agents: From theory to practice. *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, CA, US*, pages 312–319, 1995.

[183] Anand Rao and Michael Georgeff. Modeling rational agents within a BDI-architecture. *Readings in agents*, pages 317–328, 1997.

[184] Stuart Reeves. Envisioning Ubiquitous Computing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 1573–1582, New York, NY, USA, 2012. Association for Computing Machinery.

[185] Philipp Reinecke, Katinka Wolter, and Aad van Moorsel. Evaluating the adaptivity of computing systems. *Performance Evaluation*, 67:676–693, 08 2010.

[186] Paul Rendell. A Universal Turing Machine in Conway's Game of Life. *Journal of Cellular Automata*, 8, 07 2011.

[187] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21:25—-34, 1987.

[188] Urban Richter. *Controlled self-organisation using learning classifier systems.* KIT Scientific Publishing, 2019.

[189] Urban Richter, Moez Mnif, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. Towards a generic observer/controller architecture for organic computing. *INFORMATIK 2006–Informatik für Menschen, Band 1*, 2006.

[190] Matthias Rohr, Simon Giesecke, and Wilhelm Hasselbring. A Classification Scheme for Self-adaptation Research. 2006.

[191] Daniel Ronzani. The battle of concepts: Ubiquitous Computing, pervasive computing and ambient intelligence in Mass Media. *Ubiquitous Computing and Communication Journal*, 4(2):9–19, 2009.

[192] Stanley J. Rosenschein and Leslie Pack Kaelbling. The synthesis of digital machines with provable epistemic properties. In *Theoretical aspects of reasoning about knowledge*, pages 83–98. Elsevier, 1986.

[193] Stefan Rudolph, Jan Kantert, Uwe Jänen, Sven Tomforde, Jörg Hähner, and Christian Müller-Schloer. Measuring Self-Organisation Processes in Smart Camera Networks. In Ana Lucia Varbanescu, editor, *Proceedings of the 29th International Conference on Architecture of Computing Systems (ARCS 2016)*, chapter 14, pages 1–6. VDE Verlag GmbH, Berlin, Offenbach, DE, Nuremberg, Germany, April 2016.

[194] Stefan Rudolph, Sven Tomforde, and Jörg Hähner. A mutual influence-based learning algorithm. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016), Volume 1, Rome, Italy, February 24-26, 2016*, pages 181–189. SciTePress, 2016.

[195] Stefan Rudolph, Sven Tomforde, and Jörg Hähner. Mutual influence-aware runtime learning of self-adaptation behavior. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 14(1):1–37, 2019.

[196] Stefan Rudolph, Sven Tomforde, Bernhard Sick, and Jörg Hähner. A mutual influence detection algorithm for systems with local performance measurement. In *2015 IEEE 9th International Conference on Self-Adaptive and Self-Organizing Systems*, pages 144–149. IEEE, 2015.

[197] Stuart J. Russell. *Artificial intelligence a modern approach.* Pearson Education, Inc., 2010.

[198] Alejandro Salado and Roshanak Nilchiani. The Concept of Problem Complexity. *Procedia Computer Science*, 28:539–546, 12 2014.

[199] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *TAAS*, 4, 01 2009.

[200] Hartmut Schmeck. Organic computing-a new vision for distributed embedded systems. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 201–203. IEEE, 2005.

[201] Thomas Schreiber. Measuring Information Transfer. *Physical review letters*, 85:461–4, 08 2000.

[202] Bernard Scott. Cybernetics for the Social Sciences. *Brill Research Perspectives in Sociocybernetics and Complexity*, 2:1–128, 04 2021.

[203] Ivan V. Sergienko, Valeriy K. Zadiraka, Oleg M. Lytvyn, Ivan V. Sergienko, Valeriy K. Zadiraka, and Oleg M. Lytvyn. Theories of Computational Complexity. *Elements of the General Theory of Optimal Algorithms*, pages 29–73, 2021.

[204] Olivier Serrat. *The Five Whys Technique*, pages 307–310. Springer Singapore, Singapore, 2017.

[205] Cosma Rohilla Shalizi. *Causal architecture, complexity and self-organization in the time series and cellular automata.* PhD thesis, University of Wisconsin–Madison, 2001.

[206] Thomas B Sheridan, William L Verplank, and TL Brooks. Human/computer control of undersea teleoperators. In *NASA. Ames Res. Center The 14th Ann. Conf. on Manual Control*, 1978.

[207] Donghee Shin. The effects of explainability and causability on perception, trust, and acceptance: Implications for explainable AI. *International Journal of Human-Computer Studies*, 146:102551, 2021.

[208] Bernhard W. Silverman. *Density Estimation for Statistics and Data Analysis.* Routledge, 2018.

[209] Herbert A Simon. The architecture of complexity. *Proceedings of the American philosophical society*, 106(6):467–482, 1962.

[210] Taranjeet Singh and Sammer Kandpal. Overview of Autonomic Computing Systems. *International Journal of Scientific & Engineering Research*, 8:1139 – 1142, 07 2017.

[211] Michael Sipser. Introduction to the Theory of Computation. *ACM Sigact News*, 27(1):27–29, 1996.

[212] Jacopo Soldani and Antonio Brogi. Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. *ACM Computing Surveys (CSUR)*, 55(3):1–39, 2022.

[213] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546*, 2017.

[214] Matthias Sommer, Sven Tomforde, and Jörg Hähner. An organic computing approach to resilient traffic management. In *Autonomic Road Transport Support Systems*, pages 113–130. Springer, 2016.

[215] Russell K. Standish. Concept and definition of complexity. In *Intelligent complex adaptive systems*, pages 105–124. IGI Global, 2008.

[216] Mireille Sthijns, Vanessa LaPointe, and Clemens Blitterswijk. Building Complex Life Through Self-Organization. *Tissue Engineering Part A*, 25, 08 2019.

[217] Emilio Sulis and Kuldar Taveter. *Agents and Organization Studies*, pages 37–50. Springer International Publishing, Cham, 2022.

[218] Zetian Sun and Bowen Xu. A flocking algorithm of multi-agent systems to optimize the configuration. In *2021 China Automation Congress (CAC)*, pages 7680–7684. IEEE, 2021.

[219] Stefan Taranu and Jens Tiemann. On assessing self-adaptive systems. In *2010 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 214–219. IEEE, 2010.

[220] David Tennenhouse. Proactive computing. *Communications of the ACM*, 43, 05 2000.

[221] Helmut Thome. *Zeitreihenanalyse: Eine Einführung für Sozialwissenschaftler und Historiker*. R. Oldenbourg Verlag, 2005.

[222] Sven Tomforde and Martin Goller. To Adapt or Not to Adapt: A Quantification Technique for Measuring an Expected Degree of Self-Adaptation. *Computers*, 9:21, 03 2020.

[223] Sven Tomforde and Jörg Hähner. Organic Network Control: Turning Standard Protocols into Evolving Systems. *Biologically Inspired Networking and Sensing: Algorithms and Architectures*, pages 11–35, 01 2011.

[224] Sven Tomforde, Jörg Hähner, and Bernhard Sick. Interwoven Systems. *Inform. Spektrum*, 37(5):483–487, 2014.

[225] Sven Tomforde, Uwe Jänen, Jörg Hähner, and Martin Hoffmann. SmaCCS: Smart camera cloud services: Towards an intelligent cloud-based surveillance system. In *Proceedings of the 10th International Conference on Informatics in Control, Automation, and Robotics (ICINCO'13), INSTICC 2013*, pages 288 – 293, 07 2013.

[226] Sven Tomforde, Jan Kantert, and Bernard Sick. Measuring Self-organisation at Runtime - A Quantification Method based on Divergence Measures. In *International Conference on Agents and Artificial Intelligence*, pages 96–106. SciTePress, 01 2017.

[227] Sven Tomforde and Christian Müller-Schloer. Incremental Design of Adaptive Systems. *J. Ambient Intell. Smart Environ.*, 6(2):179–198, 2014.

[228] Sven Tomforde, Yanneck Ohl, and Ingo Thomsen. Incident-Aware Distributed Signal Systems in Self-Organised Traffic Control Systems. In *Proceedings of the 9th International Conference on Vehicle Technology and Intelligent Transport Systems, VEHITS*, pages 15–26, 2023.

[229] Sven Tomforde, Holger Prothmann, Jürgen Branke, Jörg Hähner, Moez Mnif, Christian Müller-Schloer, Urban Richter, and Hartmut Schmeck. Observation and Control of Organic Systems. In Christian Müller-Schloer, Hartmut Schmeck, and Theo Ungerer, editors, *Organic Computing - A Paradigm Shift for Complex Systems*, pages 325–338. Springer, 2011.

[230] Sven Tomforde, Holger Prothmann, Fabian Rochner, Jürgen Branke, Jörg Hähner, Christian Müller-Schloer, and Hartmut Schmeck. Decentralised Progressive Signal Systems for Organic Traffic Control. In *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2008, 20-24 October 2008, Venice, Italy*, pages 413–422, 2008.

[231] Sven Tomforde, Stefan Rudolph, Kirstie Bellman, and Rolf P. Würtz. An Organic Computing Perspective on Self-Improving System Interweavin, at runtime. In *2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016*, pages 276–284. IEEE Computer Society, 2016.

[232] Sven Tomforde, Bernhard Sick, and Christian Müller-Schloer. Organic Computing in the Spotlight. *CoRR*, abs/1701.08125, 2017.

[233] Ichiro Tsuda, Yutaka Yamaguti, and Hiroshi Watanabe. Self-Organization with Constraints—A Mathematical Model for Functional Differentiation. *Entropy*, 18:74, 02 2016.

[234] J.P.G. van Brakel. Robust peak detection algorithm using z-scores. `https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362`. Accessed: 2022-05-12.

[235] Norha Villegas, Hausi Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. A Framework for Evaluating Quality-Driven Self-Adaptive Software Systems. *SEAMS 2011*, 05 2011.

[236] Vito Volterra. Leçons sur la théorie mathématique de la lutte pour la vie (paris: Gauthier-villars, 1931). *seq. Volterra148Leçons sur la théorie mathématique de la lutte pour la vie1931*, page 148, 1976.

[237] Roy Want, Trevor Pering, and David Tennenhouse. Comparing autonomic and proactive computing. *IBM Systems Journal*, 42:129 – 135, 02 2003.

[238] Mark Weiser. Ubiquitous Computing. *Computer*, 26(10):71–72, oct 1993.

[239] Gerhard Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press, 1999.

[240] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 2 edition, September 2000.

[241] Danny Weyns. *An Introduction to Self?Adaptive Systems: A Contemporary Software Engineering Perspective*. John Wiley & Sons, 2021.

[242] Danny Weyns and Michael Georgeff. Self-Adaptation Using Multiagent Systems. *IEEE Software*, 27:86–91, 01 2010.

[243] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, 2019.

[244] Karoline Wiesner and James Ladyman. Measuring complexity. *arXiv*, September 2019. 10 pages.

[245] Paul Wilson, Larry Dell, and Gaylor Anderson. Root Cause Analysis: A Tool for Total Quality Management. *Journal For Healthcare Quality*, 18:40, 01 1996.

[246] Stephen Wolfram. *Thermodynamics and Hydrodynamics of Cellular Automata*, pages 259–265. crc Press, 03 2018.

[247] Michael Wooldridge. *An Introduction to MultiAgent Systems (2. ed.)*. John wiley & sons, 2009.

[248] W. Andy Wright, Robert E. Smith, Martin Danek, and Pillip Greenway. A generalisable measure of self-organisation and emergence. In *International Conference on Artificial Neural Networks*, pages 857–864. Springer, 2001.

[249] Hu-Sheng Wu. A survey of research on anomaly detection for time series. In *2016 13th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, pages 426–431. IEEE, 2016.

[250] Micaela Wuensche, Sanaz Mostaghim, Hartmut Schmeck, Timo Kautzmann, and Marcus Geimer. Organic computing in off-highway machines. In *Proceedings of the second international workshop on Self-organizing architectures*, pages 51–58, 2010.

[251] Jochen Wuttke, Yuriy Brun, Alessandra Gorla, and Jonathan Ramaswamy. Traffic routing for evaluating self-adaptation. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 27–32, 06 2012.

[252] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V Chawla. A deep neural network for unsupervised anomaly detection and diagnosis in multivariate time series data. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 1409–1416, 2019.

[253] Kai Zhang, Dahai Han, and Hongping Feng. Research on the complexity in Internet of Things. In *2010 International Conference on Advanced Intelligence and Awarenss Internet (AIAI 2010)*, pages 395 – 398, 11 2010.

[254] Keli Zhang, Marcus Kalander, Min Zhou, Xi Zhang, and Junjian Ye. An influence-based approach for root cause alarm discovery in telecom networks. In *International Conference on Service-Oriented Computing*, pages 124–136. Springer, 2020.

[255] Yan Zhang, Anna Liu, and Wei Qu. Software architecture design of an autonomic system. In *Proc of 5th Australasian Workshop on Software and System Architectures, Melbourne, Australia*, pages 5–11, 2004.

[256] Yingying Zhang, Zhengxiong Guan, Huajie Qian, Leili Xu, Hengbo Liu, Qingsong Wen, Liang Sun, Junwei Jiang, Lunting Fan, and Min Ke. CloudRCA: A root cause analysis framework for cloud computing platforms. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 4373–4382, 2021.

[257] AMSA - Global Maritime Distress and Safety System. `https://www.amsa.gov.au/safety-navigation/navigation-systems/global-maritime-distress-and-safety-system-gmdss-radio-logbook`. Accessed: 2023-06-18.

[258] AUTOMOBILES: RACES; Everybody Manages to Win Something At the Daytona Beach Contests. http://select.nytimes.com/gst/abstract.html?res=F60C16FD355A137A93C6AB1788D85F438585F9. Accessed: 2022-10-01.

[259] Covid Contact Tracing in Germany. https://www.dw.com/de/gesundheits%C3%A4mter-mit-papier-stift-und-fax-gegen-corona/a-56347106. Accessed: 2023-07-08.

[260] Danish Maritime Authority Homepage. https://dma.dk. Accessed: 2023-04-18.

[261] DMA AIS Data. http://web.ais.dk/aisdata/. Accessed: 2023-04-18.

[262] Encyclopaedia Britannica: Ctesibius Of Alexandria. https://www.britannica.com/biography/Ctesibius-of-Alexandria. Accessed: 2022-10-01.

[263] Github Repository with the Source Code for this Thesis. https://github.com/Inumeg/saso. Accessed: 2023-06-01.

[264] International Maritime Organization, AIS. https://www.imo.org/en/OurWork/Safety/Pages/AIS.aspx. Accessed: 2023-04-18.

[265] Internet Domain Survey. https://www.isc.org/survey/. Accessed: 2022-10-01.

[266] Merriam Webster Online Dictionary - Complex. https://www.merriam-webster.com/dictionary/complex. Accessed: 2022-10-01.

[267] More Auto Computers Means More Complicated, Costly and Longer Repairs. https://ceinetwork.com/news-archive/auto-computers-means-complicated-costly-longer-repairs/. Accessed: 2022-10-01.

[268] Nord-Ostsee-Kanal. https://www.gdws.wsv.bund.de/DE/wasserstrassen/01{_}bundeswasserstrassen/Kueste/NOK.html?nn=1214418. Accessed: 2023-04-18.

[269] The Scrum guide. https://scrumguides.org/scrum-guide.html. Accessed: 2022-10-01.

[270] The CAS Wiki: Self-Star Properties. http://wiki.cas-group.net/index.php?title=Self-Star_Properties. Accessed: 2022-10-01.

[271] XKCD - Health Data. https://xkcd.com/2620/. Accessed: 2023-04-18.

[272] Google Maps. https://www.google.de/maps/. Accessed: 2023-04-18.

Appendix A

Formalities

A.1 Author's Declarations

With regard to the present dissertation, I declare the following:

Independence: Apart from the advice of my supervisor, this dissertation was written by me independently in terms of content and form. Only the indicated auxiliary materials and tools were used.

Publications: This paper has not been submitted in whole or in part to any other body for an examination procedure. The work is partly based on previously published articles. These are listed in Chapter 1.4. There are no further publications or submissions for publication.

Compliance with rules: The thesis was written in compliance with the rules of good scientific practice of the Deutsche Forschungsgemeinschaft.

Withdrawal of a degree: No academic degree was withdrawn from me.

Jürgen-Martin Goller

A.2 Erklärungen des Autors

Bezüglich der vorliegenden Dissertationsschrift erkläre ich Folgendes:

Eigenständigkeit: Die vorliegende Abhandlung wurde – abgesehen von der Beratung durch meinen Betreuer – nach Inhalt und Form von mir eigenständig verfasst. Es wurden nur die angegebenen Hilfsmittel verwendet.

Veröffentlichungen: Die vorliegende Abhandlung wurde weder in Gänze noch in Teilen einer anderen Stelle im Rahmen eines Prüfungsverfahrens vorgelegt. Die Arbeit basiert zu Teilen auf bereits veröffentlichten Beiträgen. Diese sind in Kapitel 1.4 aufgeführt. Weitere Veröffentlichungen oder Einreichungen zur Veröffentlichung liegen nicht vor.

Regeleinhaltung: Die Arbeit ist unter Einhaltung der Regeln guter wissenschaftlicher Praxis der Deutschen Forschungsgemeinschaft entstanden.

Entziehung eines Grades: Mir wurde kein akademischer Grad entzogen.

Jürgen-Martin Goller