University of Regensburg
Faculty of Informatics and Data Science
Chair of AI in IT-Security

# Identification of collaborative patterns in event log data using Process Mining

Master's Thesis

Thesis submitted in partial fulfillment for the requirements of the degree
"Master of Science with Honors (M.Sc. (hon))" in Management Information Systems
at the Faculty of Informatics and Data Science
University of Regensburg

Submitted to: Prof. Dr. Maria Leitner

Regensburg, 2024

Submitted by:

Benjamin Meier
Koenigswiesenweg 28
93051 Regensburg
Student ID: 2326517

# Acknowledgements

At the beginning of this thesis, I would like to express my sincere gratitude to my supervisor, Prof. Dr. Maria Leitner, who has constantly supported me throughout the entire thesis with valuable advice and interesting discussions.

Furthermore, I would like to express my heartfelt thanks to those involved in the "Honors" program at the University of Regensburg. Thanks to their constant support, I can look back on a very pleasant and rewarding Master's program. Special thanks go to Prof. Dr. Michael Dowling, who has shaped the Honors Program like no other, and to my mentor Ulrich Heckenberger, who has supported me with constant advice throughout my Master's studies. I would also like to thank all the friends I made at the University of Regensburg for the unforgettable times.

Last but not least, I would like to thank my girlfriend Christina for her tireless support and understanding during the writing of this thesis, but also throughout my entire academic journey.

Benjamin Meier

# Abstract

The use of collaborative robots ("Cobots") enables companies to open up new possibilities in manufacturing, such as the ability to customize products more easily, reduce costs, or deploy workers more efficiently. Collaborative Patterns categorize and describe the work between humans and cobots. The aim of this work is to analyze the current state of research on Human-Robot Collaboration (HRC) and to automatically recognize Collaborative Patterns between different actors in event logs.

For this purpose, a systematic literature search was carried out in common scientific databases. Furthermore, a mining algorithm for recognizing Collaborative Patterns was designed, developed, implemented in Python and evaluated using predefined use cases.

During the research, it was found that there is no uniform definition of patterns, but a particularly common definition ("Coexistence", "Synchronized", "Cooperation", "Collaboration"). Recognition of these patterns was therefore implemented in the algorithm.

The developed algorithm allows the successful recognition of patterns in an event log and can assign corresponding patterns to individual activities within a case.

The limitations of the algorithm lie in the lack of ability to separate activities in terms of space, which would be advantageous for some patterns. However, this is caused by data availability issues in traditional event logs. A possible remedy is the use of generative AI or the collection of additional positioning data for the event log.

The work underlines the importance of research in the field of HRC and, in particular through the development of the algorithm, makes a further contribution to a better understanding of how different actors work together in a production scenario.

# Kurzfassung

Der Einsatz von kollaborativen Robotern ("Cobots") ermöglicht es Unternehmen, neue Möglichkeiten in der Produktion zu erschließen und so beispielsweise Produkte leichter zu individualisieren, Kosten zu senken oder Mitarbeiter sinnvoller einzusetzen. Kollaborative Muster kategorisieren und beschreiben die Zusammenarbeit zwischen Mensch und Cobot. Ziel dieser Arbeit war es, den aktuellen Stand der Forschung zur Mensch-Roboter-Kollaboration zu analysieren und kollaborative Muster zwischen verschiedenen Akteuren automatisiert in Event Logs zu erkennen.
Dazu wurde eine systematische Literaturrecherche in gängigen wissenschaftlichen Datenbanken durchgeführt. Darüber hinaus wurde ein Mining-Algorithmus zur Erkennung kollaborativer Muster konzipiert, entwickelt, in Python implementiert und anhand vordefinierter Anwendungsfälle evaluiert.
Im Rahmen der Recherche wurde festgestellt, dass es keine einheitliche Definition von Mustern gibt, jedoch eine besonders verbreitete Definition („Coexistence", „Synchronized", „Cooperation", „Collaboration"). Die Erkennung dieser Muster wurde daher in den Algorithmus implementiert.
Der entwickelte Algorithmus ermöglicht eine erfolgreiche Erkennung von Mustern in einem Event Log und kann einzelnen Aktivitäten innerhalb eines Falls entsprechende Muster zuordnen.
Die Grenzen des Algorithmus liegen in der fehlenden Fähigkeit, Aktivitäten räumlich zu trennen, was für einige Muster von Vorteil wäre. Dies ist jedoch durch die in klassischen Event Logs zur Verfügung stehenden Daten bedingt. Eine mögliche Abhilfe ist die Verwendung von generativer KI oder die Sammlung zusätzlicher Positionsdaten für den Event Log.
Die Arbeit unterstreicht die Bedeutung der Forschung auf dem Gebiet der Mensch-Roboter-Kollaboration und leistet insbesondere durch die Entwicklung des Algorithmus einen weiteren Beitrag zum besseren Verständnis der Zusammenarbeit verschiedener Akteure in einem Produktionsszenario.

# Contents

# List of Figures

# List of Tables

# Listings

# List of Abbreviations

| | |
|---|---|
| IS | Information Systems |
| GUI | Graphical User Interface |
| HRC | Human-Robot Collaboration |
| SLR | Systematic Literature Review |
| AI | Artificial Intelligence |
| AR | Augmented Reality |
| BPMN | Business Process Modeling Notation |
| Llm | Large Language Model |

# Chapter 1

# Introduction

The use of robots in production is becoming increasingly important as they enable significant efficiency gains and cost savings [1]. Their ability to perform tasks precisely and repeatably helps to increase product quality and reduce errors. Robots can work around the clock, increasing production capacity and improving flexibility in manufacturing [2]. They are also able to perform hazardous or ergonomically demanding tasks, which improves the safety and well-being of employees. With advances in automation technology and increasing global competitive pressure, the use of robots in production is becoming more and more essential [1]. Collaborative robots, or cobots, are designed to work safely and efficiently side-by-side with humans, without physical barriers or safety fences. These robots promote flexibility in production by taking over simple and repetitive tasks, allowing human workers to focus on more complex and value-adding activities.

## 1.1 Motivation

Companies often find it difficult to identify or model the exact processes within their production. Although process mining (see section 2.3) has significantly simplified process recognition in recent years, there are still a number of unanswered questions, particularly in the collaboration between human operators and robots or cobots.
In recent years, literature has uncovered a number of "Collaborative Patterns", also sometimes called "Collaborative scenarios". These patterns define the way of collaboration between human actors and robotic actors. Understanding the actual form of collaboration between the existing actors within a production system has a number of benefits for companies, including:

- **Optimization of work processes**: If it is known how operators and robots interact, workflows can be optimized. For example, the sequence of operations between activities can be optimized to reduce the number of work handovers between operators [2].

- **Optimization of resource allocation**: Proper understanding and categorization of tasks can help allocate resources (human and robot) more efficiently. For example, tasks that require delicate manipulation might be better suited for humans, while repetitive or heavy tasks could be assigned to robots. It also helps to understand bottlenecks caused by one actor waiting for the results of another actor's work.
- **Improved risk assessment**: Different scenarios involve different interactions between humans and robots. Understanding these can help in assessing risks and implementing necessary safety measures [3]. In some cases, such a risk assessment may even be required by regulatory authorities.
- **Improved ergonomics**: Understanding the way humans and robots work together helps in designing more ergonomic workplaces to reduce the strain on human workers.

## 1.2 Research Questions

Considering the outlined relevance of identifying Collaborative Patterns in production environments, this work aims to investigate such patterns in the domain of HRC and how Collaborative Patterns can be reliably detected from a production event log. In order to achieve this, the following research questions will be answered:

RQ1: "How can we reliably identify Collaborative Patterns in event log data using Process Mining?"

RQ2: "What are the main challenges in identifying Collaborative Patterns in event log data?"

The first research question, RQ1, asks the question on how Collaborative Patterns can be identified in event logs, which will be answered by developing an algorithm. Research question RQ2 concerns the challenges that complicate the identification of Collaborative Patterns. This work contributes to the expanded knowledge base by conducting a systematic analysis of the literature to date, in addition to creating a theoretical algorithm for detecting Collaborative Patterns. Furthermore, this algorithm will also be implemented and evaluated.

## 1.3 Structure of the Work

The organization of this work is as follows:
Chapter 2 lays out the theoretical background regarding the rest of this work. Chapter 3 describes the procedure in which the Systematic Literature Review (SLR) was conducted, including the research method, search strings and literature databases used. Chapter 4 lays out the results of the SLR, following a predefined classification scheme. The next chapter, 5, defines practice-oriented use cases which will be used to evaluate the algorithm in a later chapter. Additionally, the procedure of generating artificial event logs is described. Chapter 6 describes the theoretical development of the algorithm used to identify Collaborative Patterns, including assumptions about the event log as well as target classifications for certain workflows. Additionally, a high-level description of the algorithm itself is present in this chapter. The next chapter 7 implements the theoretical algorithm in Python and describes setup, implementation and usage of the program. Likewise, chapter 8 describes the implementation of a log generator that was also implemented during this work. Chapter 9 then applies the implemented algorithm on a number of event logs in order to evaluate the quality of classification. Additionally, a number of additional perspectives are described, such as runtime analysis, limitations and future improvements. Finally, chapter 10 concludes this work.

# Chapter 2

# Theoretical Background

## 2.1 Cobots

Collaborative robots, or "Cobots," have significantly impacted manufacturing scenarios in recent years. Unlike traditional industrial robots, cobots are designed to work alongside human workers, increasing productivity and safety in manufacturing environments [4]. The primary goal of cobots is to assist with tasks that are repetitive, dangerous, or require precision, allowing human workers to focus on more complex and creative aspects of production. Cobots are typically equipped with advanced sensors and software that enable them to sense and respond to their environment, ensuring safe interaction with humans [5]. Cobots in production can be utilized in various applications, such as:

- **Assembly**: Cobots assist in assembling products by performing a number of activities such as handling small parts, applying adhesives, and performing precise movements.
- **Quality Inspection**: If equipped with vision systems, cobots can perform detailed inspections of products, ensuring consistent quality and freeing human workers to address issues that require problem-solving skills.
- **Material Handling**: Cobots efficiently handle the transportation of materials within a production facility, even outside of limited and fenced areas.

There are several advantages to integrating cobots into production scenarios:

- **Increased Productivity**: By automating repetitive and time-consuming tasks, cobots help increase overall production rates [6].
- **Improved Safety**: Cobots can take over dangerous tasks, reducing the physical strain on workers and risk of workplace injuries.
- **Cost Efficiency**: Cobots can reduce labor costs and minimize errors, resulting in significant cost savings over time.

## 2.2 Collaboration Scenarios

"Collaboration" generally describes the cooperative work of different actors to achieve a common goal. Collaboration is not limited to work between humans and cobots, but regularly occurs in various scenarios, for example in production, software development or other office work. It is important to differentiate collaboration scenarios (also called "Collaboration patterns" or "Collaborative patterns"), as there are a number of different ways of collaboration between both human and non-human (robotic) actors. This can be done by identifying collaboration patterns with distinct characteristics. There is currently no generally recognized definition of collaborative scenarios in research, however, a large number of publications ([7], [8], [9], [10], [11], [12],[13] [14], [15], [16] [17], [18], [19] and [20]) follow a common definition of collaboration scenarios. The most commonly defined scenarios are the following:

- **Coexistence**: The human operator and the cobot work independently in a shared environment, without interacting with each other.
- **Synchronized**: The human operator and the cobot work on the same workpiece during a process, but at different times.
- **Cooperation**: The human operator and the cobot work simultaneously on the same workpiece, but on different areas of the workpiece.
- **Collaboration**: The human operator and the cobot work interactively ´towards the same goal on the same workpiece (simultaneously).

In addition, there are a number of different definitions. These are not considered in this thesis for the purpose of identifying patterns of collaboration. The exact definition of collaborative scenarios from scientific literature will be described in more detail in section 4.3, more precisely in 4.3.1 and 4.3.2.

## 2.3 Process Mining

Few process management techniques have been as influential in recent years as Process Mining. Originally proposed by W.M.P. van der Aalst [21], it quickly became one of the largest fields of research and spawned a variety of software tools and companies dedicated to Process Mining. The main goal of Process Mining is to analyze and improve business processes. The motivation for this discipline comes from the fact that organizations use a large number of workflow management software and other software systems in isolation, often ignoring the interactions between these applications. Typically, such tools offer no way to diagnose or monitor ongoing workflows [21]. The central idea of Process Mining is to analyze event logs generated by enterprise systems or applications in order to gain insights into the underlying "actually executed" processes. This area is not limited to software systems, but can be applied to "process, control, data, organizational and social structures" [22, p.512]. Process Mining collects and analyzes the "digital traces" of processes in order to gain insights into the process flow. Process Mining can be divided into three areas [23]:

- **Process discovery**: This area involves the actual discovery of the process underlying a particular event log without using any prior information. This is achieved by using one of the many available Process Mining algorithms such as the inductive mining algorithm. Such algorithms take event logs as input and create a Petri net that explains the behavior recorded in the event log.

- **Process conformance checking**: The second area of Process Mining is Conformance checking. In this step, the previously created process model is compared with the event logs of the modeled process. The aim of this approach is to check whether the model matches the reality as recorded in the logs. It also shows whether rules, such as additional management confirmations for high-impact transactions, are actually followed. Another advantage of this step is that it is possible to "detect, localize and explain deviations and measure the severity of these deviations" [23, p.33]. This functionality of Process Mining has triggered further research into algorithms for checking conformance, such as [24].

- **Process enhancement**: The third area of Process Mining is process enhancement. As the name suggests, it is about building on the process models discovered in the previous steps to enhance these existing processes. Unlike the previous step, conformance checking, which merely confirms the match between model and reality, this area aims to actually change the existing processes in order to improve and enhance them. Examples of this could be the elimination of discovered bottlenecks or the improvement of key figures such as processing times or service levels.

Event logs are critical for Process Mining, requiring at least a case identifier, an activity, and a timestamp, but more fields can improve model quality. The requirements for event logs in the algorithm proposed in this work go further, as described in section 6.1.

# Chapter 3

# Setup of the Systematic Literature Review

The following chapter describes the research method that was used in the SLR section of this work in detail. A structured approach is essential in order to perform an effective literature review, thus special emphasis was placed on choosing a research method that is suitable for research about Information Systems (IS). This was especially important, as a lack of properly conducted literature reviews will hinder theoretical and practical process [25].

## 3.1 Applied Research Method

The framework introduced by Yair Levy and Timothy J. Ellis in 2006 [26] was chosen as a research method. One unique advantage and the main reason why it was chosen is the explicit focus on information systems literature reviews. The method proposes a systematic approach to research consisting of three main stages:

1. **Inputs**: The initial stage of this research technique focuses on the important task of gathering high-quality literature to conduct a thorough literature review. The authors emphasize that if the initial information gathered is inaccurate or of poor quality, the subsequent outcomes will also be irrelevant, following the well-known principle of *"garbage-in/garbage-out"*. Levi and Ellis provide insights into locating reputable IS literature, ensuring the validity of search outcomes, and assessing their relevance to the research topic. A list of reputable databases and journals is presented. Additionally, they outline the proper utilization of Keyword-, Backward-, and Forward-search methods.

2. **Processing**: In the second phase of a literature review, the relevant literature identified in the previous step must be transformed into actionable information that can form the base for further research [27]. This involves understanding, analyzing, synthesizing, and applying the literature.

3. **Outputs**: The final step of the research method outlined by Levi and Ellis is concerned with the actual composition of the text. It provides guidelines for constructing arguments and discusses strategies for dealing with common problems or errors. Ultimately, it helps researchers effectively summarize their work.

## 3.2 Identification of Relevant Literature

In this section, the steps taken to gather relevant literature are laid out.

### 3.2.1 Search Strings

To identify existing literature concerning Collaborative Patterns in HRC and Process Mining, a SLR was performed. This was carried out using the following search terms:

**Term 1:**
This search term focuses on HRC itself, aiming to discover all relevant literature, but with no particular focus on Process Mining.

```
((Collaborat*)
AND (scenario* OR pattern* OR way* OR "human machine" OR "human robot"))
OR HRC
```

**Listing 3.1:** Search Term 1

**Term 2:**
This search term focuses on HRC, but also takes Process Mining into account. As this search term is more focused than Term 1, far fewer results were found.

```
Process Mining
AND
(((Collaborat*)
AND (scenario* OR pattern* OR way* OR "human machine" OR "human robot"))
OR HRC)
```

**Listing 3.2:** Search Term 2

Both search terms are searched in both the title and the "abstract" of the scientific publications, and in some databases, due to technical limitations, also the full text, as described in 3.2.2.

### 3.2.2 Literature Databases

In order to ensure a comprehensive and in-depth literature review, several literature databases were used to provide a wide and deep base of relevant scientific sources. A search using the search strings discussed in section 3.2.1 was conducted in the following quality scientific literature databases: ProQuest, Elsevier (Sciencedirect), IEEE Xplore, ACM, Springer Link, and AIS eLibrary. These databases are particularly useful for finding quality literature because they contain research articles from the top-ranked Management

Information Systems (MIS) journals. Additionally, Google Scholar and ResearchGate were searched. Since advanced searching with wildcards and a clear title/abstract/full text separation is not possible in every database, the way of searching for literature with the search string had to be adapted slightly in each database.

### 3.2.3 Procedure of the Literature Search

The procedure that was applied during the systematic literature search will be described here.

1. In the first step, the keyword search was performed in the databases laid out in subsection 3.2.2, using the search strings defined in subsection 3.2.1. The initial search yielded 78 loosely relevant results, out of which 35 were generally deemed usable. 7 further scientific publications were added to the list from various sources (including starting literature), increasing the total of relevant papers to 42.

2. In a second step, a backward search was performed on the 42 articles, divided into backward references, backward authors, and previously used keywords. The backward references search identified another 36 promising articles. Backward search was also carried out on the newly discovered results. No further relevant literature was found using the backward author search and previously used keywords.

3. In a further step of the process, the search of the knowledge base for relevant literature, the forward literature search was performed. This can be divided into two sub-steps similar to the backward literature search, the forward search for references and the forward search for authors. Unfortunately, neither the forward search for authors nor the forward search for references yielded any relevant results.

4. In the last step, the results were once again screened for relevance, which removed 33 results from the list, as they were not considered sufficiently relevant to the topic.

In the end of the process, a total number of 45 scientific publications relevant to the topic were discovered. Figure 3.1 illustrates the process visually.

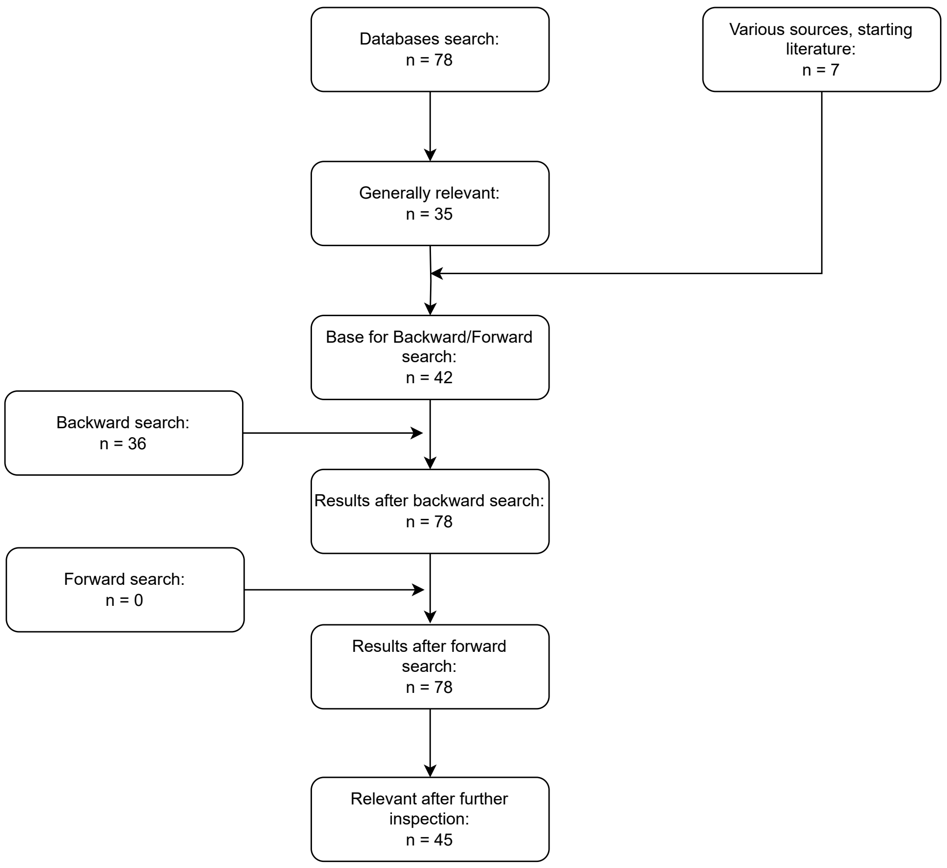

Databases search:
n = 78
Various sources, starting literature:
n = 7
Generally relevant:
n = 35
Base for Backward/Forward search:
n = 42
Backward search:
n = 36
Results after backward search:
n = 78
Forward search:
n = 0
Results after forward search:
n = 78
Relevant after further inspection:
n = 45

**Figure 3.1:** Flowchart illustrating the process of the systematic literature process

### 3.2.4 Limitations of the Systematic Literature Search

During the SLR, special emphasis was placed on careful work and identification of all publications relevant to the topic. However, any SLR has inherent limitations that affect its comprehensiveness and objectivity. In this case, the limitations are as follows:

1. **Language limitations**: The review included only papers published in English and German, excluding research in other languages. This limitation narrows the scope and may introduce language bias, potentially missing important findings from non-English and non-German sources.

2. **Handling large amounts of data**: Due to the large number of total results, it was not possible to review all of them. The review process was conducted in each database until no relevant results were found for an extended period of time. This method, although practical, may have missed relevant studies published later or not immediately identified as relevant.

3. **Limited Number of Databases**: Only a limited number of scientific databases were used for the review. This limitation may result in missing relevant studies indexed in other databases not included in the review.

4. **Access Restrictions**: Some databases could not be accessed by the author due to licensing restrictions. This limitation further reduces the scope of the review by excluding potentially important studies available in those inaccessible databases.

5. **Subjective Bias in Selection**: The selection process was based on the abstracts of papers, relying on subjective judgment to determine their relevance to the study's focus. This introduces the possibility of bias, as decisions on relevance are inherently subjective and might have been taken differently by other reviewers.

Despite these constraints, the review successfully gained a good overview of the topic, as described in subsection 3.2.3.

# Chapter 4

# Results of the Systematic Literature Review

After explaining the groundwork and research method of the systematic literature review, results are presented in chapter 4.

## 4.1 Classification Model

Based on the research questions "How can we reliably identify Collaborative Patterns in event log data using Process Mining?" and "What are the main challenges in identifying Collaborative Patterns in event log data?" (see section 1.2), the literature was thematically clustered according to the different identified topics. A division was made on the vertical axis with regard to main topics, "HRC in general" and "Collaborative Patterns". In addition, the articles of each main topic were again divided on the horizontal axis according to their subtopics. The sub chapters of the results chapter (4) are also structured according to this division. The entirety of the literature was then placed into the classification model. Consequently, this leads to the development of the following classification model, which forms the basis of this thesis:

| | **General topics** | **Literature Reviews** | **Use Cases** | |
|---|---|---|---|---|
| **HRC in general** | [11], [28], [29], [30] [31] | [32], [33], [34], [13], [35], [14] | [12], [36], [37], [38], [39], [40], [41], [42], [43], [19], [44] | |
| | **Common Definition** | **Other Definitions** | **General** | **Identification** |
| **Collaborative Patterns** | [8], [9], [10], [11] [7], [45], [12], [14] [15], [46], [16], [17] [18], [19], [20], [47] [48] | [49], [50], [51], [52], [53] | [54], [55], [56] | [57], [58] |

**Table 4.1:** Classification scheme of literature discovered in the SLR

## 4.2 Human Robot Collaboration in General

The following section 4.2 will describe results of the systematic literature review which focus on HRC in general, and not on Collaborative Patterns in specific.

### 4.2.1 General Topics about HRC

The results described in this subsection describe common groundwork in the realm of HRC, including available technologies and general considerations.
[29] is known as the first research paper to mention Cobots. The authors describe a cobot as a robotic device that collaborates with a human operator by providing virtual surfaces through steerable, passive joints instead of powered action. The passive nature makes this cobot potentially well-suited for safety-critical or high-force tasks. The paper focuses on a simple example cobot with a single steerable wheel joint, and develops two control modes called "virtual caster" and "virtual wall" control. Experimental results for this "unicycle cobot" are presented.
[28] considers general cooperation of Humans and robots in assembly processes. These hybrid human-machine assembly systems are designed to increase flexibility and reduce fixed costs, however, they must ensure worker safety. Intelligent assist devices help with keeping humans involved by reducing stress. Future research should focus on improving robustness for wider applications, enabling group cooperation, new robot programming interfaces, and redesigning robots to prioritize safety over accuracy through sophisticated control. This close collaboration would utilize the strengths of both humans and machines for more flexible and sustainable assembly processes.
[30] covers HRC in industrial settings, focusing on taxonomy, challenges, essential safety measures (such as speed and separation monitoring), relevant safety standards, and evaluation criteria. It provides an in-depth conceptual categorization of current HRC approaches in industry and research in the areas of awareness, intelligence, and compliance. Special emphasis is placed on the topic of security, which is extensively analyzed from the perspective of possible measures and the resulting challenges.
[31] conducted a total of 11 interviews with experts from both Germany and France in order to identify the most common implementation barriers to HRC. The authors related the issues to technical, economic, social, and safety dimensions that involve interrelationships and trade-offs. In order to overcome these multi-dimensional challenges and to promote effective HRC implementations that will unlock benefits such as increased productivity and optimized performance, the study proposed research directions that explore technical safety innovations and cost-effective strategies. The authors noted that focus should be put on a holistic approach that considers all dimensions together.
The article [11], published by the Frauenhofer Institute for industrial engineering (IAO), gives an overview over the current state of HRC applications in Germany, referred to as "cage-free", or "inherently safe" by the authors. The study aimed to identify companies using lightweight robots on their production lines and to learn about their experiences with

robot implementation, coworker acceptance, and operational efficiency improvements. The authors conclude that ergonomic improvements are a key reason for employing HRC. Acceptance within employees was mostly achieved by providing comprehensive information about the project to everyone involved, with one study participant noting "when people start giving the robots pet names, then you know they've accepted the technology" [11, p.28]. Additionally, the study found that in practice, the patterns "Synchronized" and "Coexistance" are most often present, but rarely the pattern "Collaboration".

### 4.2.2 Literature Reviews

During the research described in the previous chapter, a number of existing SLRs were discovered, which will be shortly described in this subsection.
The first scientific SLR in this list is [32], which focuses on the human decision making aspect of HRC. The review explores factors influencing human decision making in HRC, highlighting cognitive workload and user interface design as the most prominent. It also identifies a lack of research on social, team, and psychological safety aspects. The authors recommend future studies in realistic scenarios with different cobots/tasks beyond manufacturing and mixed subjective/objective evaluation methods for more intuitive human-robot communication. Overall, a more comprehensive understanding is needed that considers social dynamics, safety, realistic contexts, and interface innovations.
Similarly to [32], [34] reviews HRC approaches in industrial settings, focusing on the key challenges of security and intuitive programming/interaction methods. The authors cover safety standards, user interfaces such as augmented/virtual reality, commercially available HRC solutions, and prominent industrial applications. Future directions are given to enable widespread, easy-to-use HRC by addressing safety through performance optimization within constraints, transferring new intuitive interfaces to the industry, introducing adaptive inclusive solutions for diverse users, and enabling robots with cognitive skills and shared autonomy to take over complex tasks. The goal is to enable true collaboration where robots complement human capabilities with autonomy and intelligence.
[13] provides an overview of HRC in industrial environments, covering current safety standards, the state of the art in control systems and collaboration methods, and the tasks assigned to collaborative robots. During the SLR, the authors analyzed case studies from industries such as electronics and automotive. They conclude that small and medium enterprises will drive the adoption of cobots as the technology becomes more accessible. They anticipate that interdisciplinary advances from other areas of robotics will be critical to addressing ongoing HRC challenges in industrial environments.
[35] focuses on key technologies used by HRC, such as Artificial Intelligence (AI), Augmented Reality (AR) or Digital Twins, while also adressing the limitations of HRC.The authors conclude that complexity, rigidity, safety, and interfacing of the aforementioned systems are the biggest challenges faced today. Integrating in-process quality control and ensuring safety through the careful design of human-robot roles are identified as critical to the successful implementation of HRC in consistent, flexible manufacturing.

[14] has contents similar to [35], while also referencing Collaborative Patterns, collaboration levels and work roles, which further describe the relationship between the human operator and the cobot.

### 4.2.3 Use Cases

During the SLR, a special emphasis was placed on the discovery of practice-oriented use cases for HRC in manufacturing, as a number of use cases are needed as the base for chapter 5, which in turn serves as a base for chapters 6 and 9. Thus, the following subsection will introduce literature which covers use cases for HRC. Scenarios which will be utilized as example use cases will be laid out in more detail in chapter 5.
The first publication, [42], which was cited by a number of other publications, including [7], describes part of the manufacturing process of wire harnesses, called "Spot Taping". In the mentioned case, there are two cobots present, one to the right and one to the left of the shop floor worker. The human worker can work on the left area of the workpiece, while the right cobot works on the right area, and vice versa. The human can trigger the movement of the workpiece to the left or right using a foot switch.
[12] covers an exemplary scenario in the production of aircraft cabin parts. The human worker and the cobot work together at the same time, doing cooperative work such as the robot drilling holes at one area of the workpiece while the human worker places additional parts on the workpiece at another area.
The next scenario described in [37] features a simple production step in automotive manufacturing, in which the human worker first places part of a car roof in an assembly station and brings it into position. The cobot then starts to apply glue to the desired positions. After this step is finished, the worker removes the part from the assembly station and places the next part.
[36] addresses a fictional case in which a worker collaborates with a cobot to assemble a wooden box. The robot performs various tasks, such as holding the box in place or providing the required parts to the worker. As the order of operations is not fixed, a large number of possibilities for process variants exist.
[44] describes a number of industry use cases, which will be laid out here. In the first scenario, dashboards are installed in cars. The cobot places the dashboard in the required position with the help of the human worker, which guides the cobot. After the part is seated in its place, both the human and the cobot independently begin placing parts of the wire harness, as well as connecting them. Another possible usage of cobots mentioned in this scientific work involves the mounting the wheel groups in a car. The worker guides the cobot lifting heavy parts to the required position. In this case, neither human nor robot would be able to accomplish their task on their own, which means that they are mutually dependent on each other. The last scenario which is contained in [44] involves the manufacturing of refrigerators. In this production line, a cobot automatically applies insulating material to the workpiece. At the same time, the human worker inspects the refrigerator and manually corrects any imperfections.

[41] again describes a number of possible use cases for cobots in manufacturing, including the scenarios "Pick and Place" (the cobot picks up a workpiece and places it at another location or orientation), process tasks such as gluing or welding, in which the cobot moves a certain tool in a fixed path, or quality inspection, in which the cobot moves a single camera in order to inspect the quality of the work piece. All activities described here can be performed independently by the cobot, without intervention of a human worker, who can instead focus on other tasks.
[38] describes a possible workflow for a simple cooperative interaction between a human worker and a cobot, in which the human needs to insert screws into holes on a workpiece, which are then tightened by the robot. The experimental setup is described in detail including the hardware used and the assumptions which were made.
Similarly, [39] describes a problem involving robot-assisted load carrying by human operators, which is a common occurrence in HRC. However, the study focuses on the technical aspects, such as counteracting the force of the human operator and taking the unknown mass of the load into account.
Like [39], the authors of [43] describe robot-assisted load carrying through hand-guiding by a human, while also providing an overview over relevant safety standards and regulations for HRC. In this example, the cobot places a workpiece for the worker, who then works on it (if necessary with the support of the cobot), and at the end the cobot takes the workpiece away again.
[40] describes yet another collaborative interaction, in which a simple box is assembled by both the human worker and a cobot. The paper focuses on the experimental technical implementation, including the usage of face recognition, eye-tracking and gesture recognition.
The scientific paper [19] describes two example cases, one concerning the assembly of seals to components, in which the robot acts autonomously. The second and more relevant case concerns the welding of metal products: a cobot works like a turntable. The initial product is placed in the cobot by a human, which then performs the steps. afterwards, the cobot places the finished product into a pallet.

## 4.3 Collaborative Patterns

The following section will cover literature which describes Collaborative patterns themselves, including their definition and identification. The following subsections will follow the structure laid out in table 4.1.

### 4.3.1 Common Definition of Collaborative Patterns

During the SLR, a large number of publications were discovered which define Collaborative Patterns in a similar way, thus, those publications will be described in this subsection. Table 4.2 gives an overview over the most common Collaborative Patterns.
The scientific papers which describe this common definition of Collaborative Patterns are

[7], [8], [9], [10], [11], [12],[13] [14], [15], [16] [17], [18], [19], [20] and [48]. Terminology was not always consistent between the existing publications. If alternative names were used by authors for patterns commonly known under another name, the alternative name was shown in the column "Alternative Names". Special care must be given with [12], as the patterns otherwise called "Cooperation" and "Collaboration" are called "Synchronized" and "Cooperating" here. The pattern otherwise called "Synchronized" is called "Autarkic" here.

| Pattern name | Alternative Names | Description | Constraints/ Dependency between actors |
|---|---|---|---|
| Cell | | The human operator and the robot work independently from each other. The robot is operated in a cage, thus no genuine cooperation scenario is present. | None |
| Coexistence | Closed [12], Independent [15], [18], [20], [47] | The human operator and the cobot work independently from each other in a common environment, usually without any physical barriers or cages. Both actors work on separate workpieces and processes, without interacting with one another. | None |
| Synchronized | Interaction [17], Autarkic [12], Sequential [20], [47], Sequential collaboration [46] | The human operator and the cobot work on the same workpiece during a process, however, they do so at different times. The steps are not independent from each other, as the output of one process step acts as an input for a subsequent process step. | Temporal dependency |
| Cooperation | Synchronized [12], Simultaneous [15], [18], [20], [47] | The human operator and the cobot work on the same workpiece concurrently, however, at different areas of the workpiece. Even though the cobot operates independently regarding time and tasks, it must still respect the worker's space. | Spacial dependency |
| Collaboration | Cooperating [12], Supportive [15], [20], [47], Assisted [18], Responsive Collaboration [46] | The human operator and the cobot work interactively on the same process on the same workpiece (concurrently). Actions of one actor depend on the actions of the other actor, meaning that a task cannot be completed by one actor alone. | Temporal, spacial dependency |

**Table 4.2:** The common definition of Collaborative Patterns

Figure 4.1 visualizes the different Collaborative Patterns regarding their respective temporal and spacial separation, based on [12, 7].

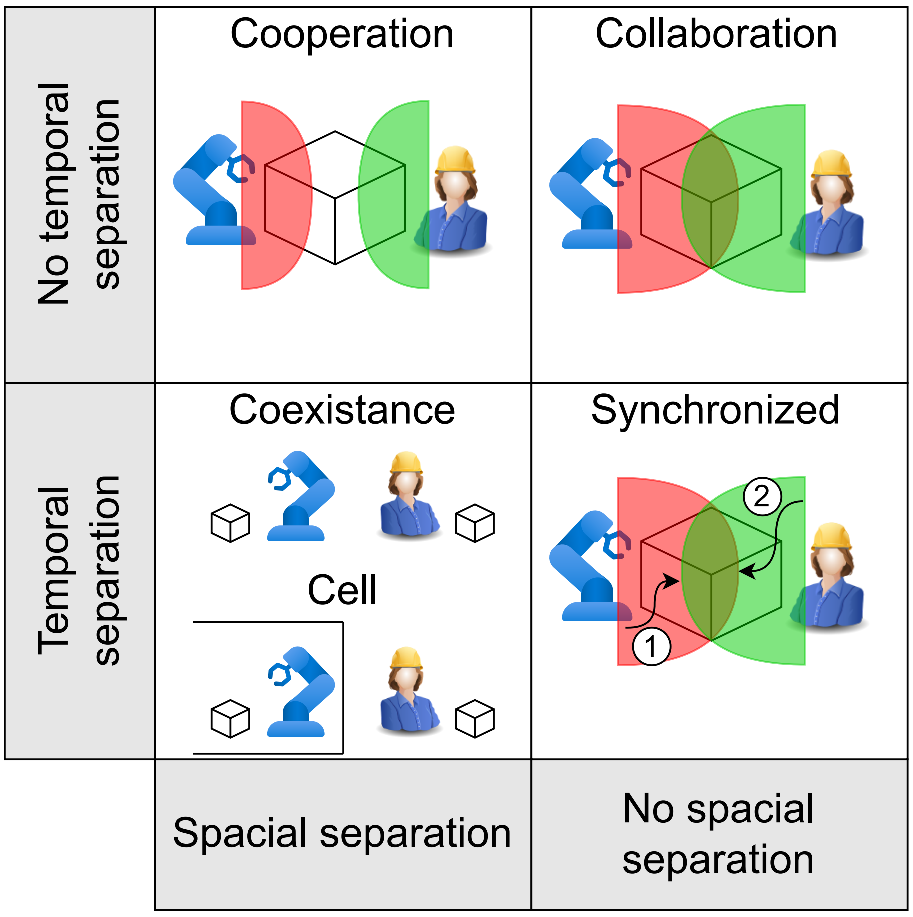


**Figure 4.1:** Common Collaborative Patterns (own illustration)

### 4.3.2 Alternative Definitions of Collaborative Patterns

In addition to the definitions described in 4.3.1, alternative definitions of Collaborative Patterns were found in the course of SLR. These and their differences to common versions are explained in more detail in the following subsection.

[49] provides four collaborative scenarios, namely sole human operation, robot support, flexible workstation (F-WS) support, and combined robot-FWS support. As the name implies, sole human operation does not involve any robot, which means that no cooperation is present. Robot support describes any case of the robot working with the human, including the traditional Synchronized, Cooperation and Collaboration patterns. Flexible workstation (F-WS) support describes a similar scenario, however, instead of a robot, the human operator is supported by appropriate equipment present on the workstation itself. Combined robot-FWS support has both a robot and a flexible workstation present, enabling all traditional patterns once again. Due to the unspecific description of collaboration in this paper, the patterns defined here will no longer be used in the further course of this work.

The authors of [50] describe the levels of "Coexistence", "Interaction", and "Collaboration". Generally, Coexistence has the same meaning as in the common definition, meaning the usage of a shared space, but no other spacial or temporal dependencies between the actors. Interaction also implies Coexistence, however, it also involves the communication between the two entities, In essence, interaction refers to the circumstance when one actor provides information and another actor responds appropriately. Collaboration, which also implies Interaction and Coexistence, again describes a scenario where human and robot (concurrently) help each other in the same task.
[51] uses a similar description for the levels of collaboration, namely "supportive", "collaborative" and "cooperative". There is no equivalent of the non-cooperative scenario of "Coexistence" or "Cell". "Supportive" interactions categorize scenarios in which the robot is not essential to the primary task itself, but instead assists the human by providing tools, materials, and information to improve the human's task performance or achieve his or her goals. In "collaborative" interactions, humans and robots work together on a task, dividing the work so that each party completes the parts best suited to their abilities. They typically interact by taking turns and passing parts or tools to each other. This form of interaction is similar to the "Synchronized" Pattern. "Cooperative" interactions extend cooperative manipulation to include force-based interactions with humans. Unlike cobots, in these interactions the robot acts as an independent agent rather than a passive assistant. The human and robot work either in direct physical contact or indirectly through a shared object, maintaining continuous and cooperative control of the task. This way of interaction is common to the pattern "Collaboration", although the robot is mainly autonomous.
[52] takes a different approach to the classification of human-robot collaboration. The authors created three classification schemes, one with the number of agents, one with the role (inactive, supporting or active) and one alignment of human activities with the nominal process definition. However, the authors did not assign specific names to the individual scenarios in the first two diagrams and did not further define/describe the specific collaboration, which is why this scheme is not used further in this thesis.
[53] does not directly describe HRC, but rather collaborative work between any actors in general. In their work, the authors describe 13 typical collaboration situations, which were extracted from pragmatic studies of numerous processes. Unfortunately, the authors do not describe the similarities between those scenarios, but rather their modelling modalities.

### 4.3.3 General Topics about Collaborative Patterns

[55] notes that the development of complex systems is increasingly collaborative, with strategies that depend on the development context at different stages. To support this, collaboration process patterns are proposed for defining, reusing, and implementing collaborative software development processes. This paper discusses these patterns, which are inspired by Van der Aalst's workflow patterns and described in CMSPEM, a process

modeling language from 2014 [59]. It explains the CMSPEM metamodel and focuses on two patterns: Duplicate in Sequence with Multiple Actors, Duplicate in Parallel with Multiple Actors, and Merge. The approach is illustrated by a case study on " Review a deliverable".
The authors of [56] describe the complexity of managing collaboration in multi-actor tasks due to potential changes in process context and strategies. They note that traditional process management solutions are inflexible and can't adapt to such changes. To enable flexible execution of collaborative tasks, they propose a late-binding mechanism that allows participants to choose or adapt strategies during execution. Collaboration strategies are modeled as process patterns that provide a number of ways to execute tasks in real time.
[54] surveys patterns that facilitate collaborative work, documenting optimal practices for recurring tasks among geographically dispersed groups. The paper examines the origins of patterns across a range of disciplines, including architectural design, software engineering, and human-computer interaction. It categorizes and compares various pattern-based approaches, such as collaboration in communities of practice, event-driven architectures, and business process management. It emphasizes the role of patterns in improving collaboration tools and suggests future research directions, including developing ontologies and recommending collaborative actions, with a particular focus on the potential for patterns to support virtual, computer-mediated collaborations. Unfortunately, the focus of the paper was not HRC.

### 4.3.4 Identification of Collaborative Patters

The following subsection will describe literature that focuses on the automatic detection or mining of Collaborative Patterns. Unfortunately, no such method directly focused on HRC was found.
[57] aims to discover collaboration patterns between a large number of software engineers in order to help the software development process. In order to do so, the authors generated and preprocessed an event log based of existing log files. Afterwards, a proprietary algorithm was applied on this log in order to mine the collaboration patterns. The method was evaluated using a case study. Unfortunately, this method is not applicable to HRC environments due to the different characteristics of both software development and HRC.
[58] is vastly different from [57]. It describes how Process Mining supports the discovery of single-participant business processes, but lacks methods for discovering collaboration models from distributed data from multiple participants. The authors address this problem by introducing a new technique that uses event logs of participant interactions. It first discovers individual processes using existing algorithms, and then analyzes message exchanges to combine these processes into a collaboration model. This model represents the behavior of the system and provides interaction analysis. The technique is implemented in a tool and validated by experiments in different domains.

## 4.4 Conclusion of the Results of the SLR

Concluding the current chapter about the results of the SLR, it can be observed that most scientific publications about Collaborative Patterns follow a common definition, as highlighted in section 4.3. There is, however, not a universally accepted definition in academia. In addition, the amount of literature dealing specifically with HRC is relatively small, and much of the available research is of limited relevance to this area, often only regarding specific fields such as safety or communication between actors. Generally, a sufficient number of use cases was discovered during the SLR, which will allow the usage of practice-oriented use cases for chapters 5 and 9. Regarding the identification of patterns, there is a notable gap in the literature regarding the automated classification of these patterns. This lack of appropriate literature underlines the need for an algorithmic way to easily and quickly recognize patterns in larger amounts of data.

# Chapter 5

# Use cases and Event Logs for the Algorithm

In this chapter, the first section will present specific use cases about Collaborative Patterns. The purpose of those use cases is the subsequent generation of practice-oriented artificial event log data in the second section, which will later be utilized in chapter 9 to evaluate the performance of the algorithm conceptualized in chapter 6 and implemented in chapter 7.

## 5.1 Use Cases

The following table lays out the use cases which will be utilized in the generation of artificial data.

| **Collaborative Patterns** | **Name** | **Source** |
|---|---|---|
| Coexistance | Coexistence: Pick-and-Place | [41] |
| Coexistance | Coexistence: Hole Drilling | Author's example |
| Synchronized | Synchronized: Glue Application | [37] |
| Cooperation | Cooperation: Spot Taping | [42] |
| Cooperation | Cooperation: Refrigerator Assembly | [44] |
| Cooperation | Cooperation: Aircraft Cabin Part Assembly | [12] |
| Collaboration | Collaboration: Box Assembly | [36] |
| Collaboration | Collaboration: Wheel Assembly | [44] |

**Table 5.1:** Use cases used in this work

### 5.1.1 Coexistence: Pick-and-Place

The process "Pick and Place" describes a very common task found in nearly every HRC production environment, which is thus often referenced in literature [41, 7]. This task involves the robot lifting a workpiece and relocating it to a different position and/or orientation than the starting position. In this context, the critical action is the manipulation of the workpiece itself, rather than any other (potentially workpiece-altering) activity performed on it. Common examples are the palletizing or packaging of goods, or the lifting of heavy objects which cannot be done by a human worker. In general, the objective include but are not limited to increased handling of goods, improved worker ergonomics or reduction of costs [41]. In the exemplary case in this work, the worker and the robot are largely independent from each other, both focused on their respective tasks, while also working without a physical barrier dividing them. For this reason, the collaborative Pattern "Coexistence" is used here. In figure 5.1 the workflows of the robot and the human are illustrated in a Business Process Modeling Notation (BPMN) diagram. Please note that there are two separate diagrams, as there is no connection or interdependence between the two actors.

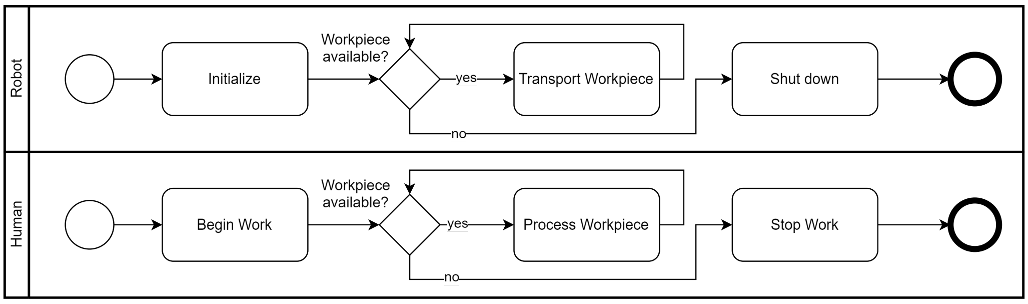


**Figure 5.1:** Pick and Place process

### 5.1.2 Coexistence: Hole Drilling

As another example of a common industrial use case is the process "Hole Drilling", which is illustrated in figure 5.2. A robot is equipped with means to drill holes, grind, polish and reposition a workpiece. This can be achieved by utilizing "end-of-arm" equipment with the robot, such as described at [60]. During the process, the robot changes automatically changes its tool to match the required process step (e.g. "Drill Hole"). In this example process, the cobot first positions the workpiece within its workspace (taking it from a box or conveyor belt) and then proceeds to drill two separate holes. The order of drilling does not matter in this case, as long as both holes are drilled before the next step, which is grinding the workpiece. Afterwards, the robot turns the workpiece around, drills another hole, polishes the workpiece and stores it, ending the process for the robot. Simultaneously to the robot performing its steps, the human worker processes different workpieces by applying a marking, not interfering with the robot at any time, which implies the pattern "Coexistence" yet again.

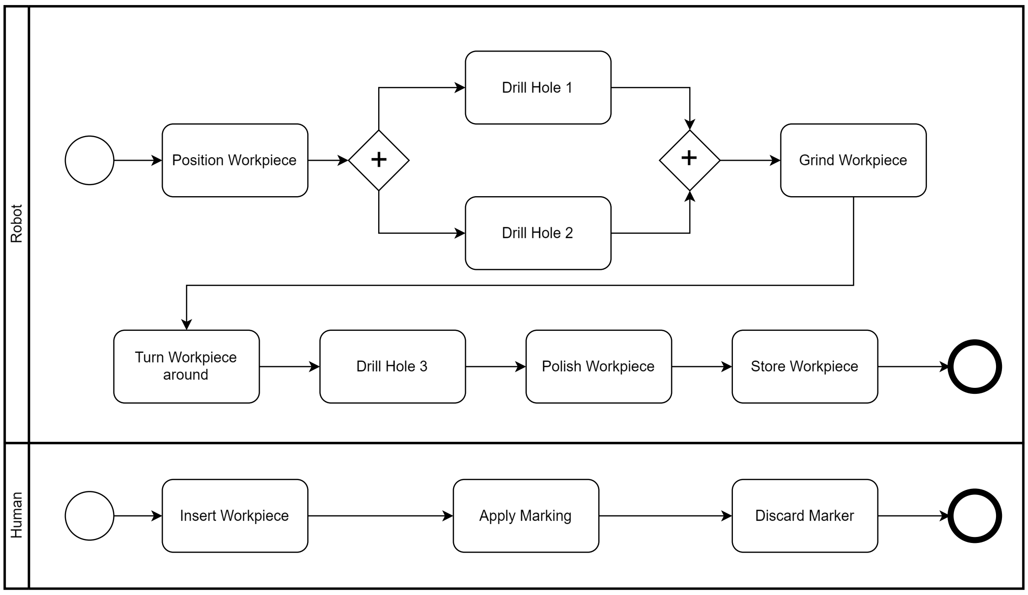


**Figure 5.2:** Hole Drilling process

### 5.1.3 Synchronized: Glue Application

The next example process, "Glue Application", was taken from [37]. In this simple scenario, a human worker takes a part (in the referenced scenario a car roof), inserts it into a fixture, and moves the part in position towards the robot. The robot then applies glue on the part, while the human worker waits. Once the robot is finished, the human removes the part from the fixture, finishing the process. As the worker is dependent on the human installing the part, and the human is dependent on the robot applying the glue, a synchronized pattern is used here.

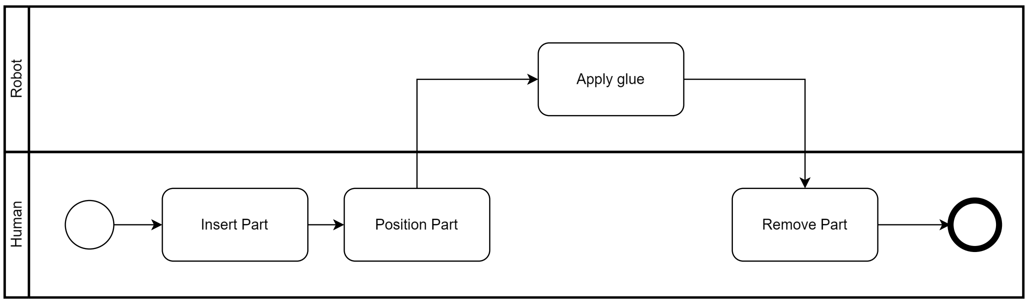


**Figure 5.3:** Glue Application process

### 5.1.4 Cooperation: Spot Taping

As already described in 4.2.3,the process of "Spot Taping" involves a human worker and two cobots which work towards a common goal, the manufacturing of wire harnesses for automotive applications. Those two cobots are placed to the right and the left of the worker, enabling them to work on the same workpiece simultaneously. The worker can move the workpiece to the left or right by utilizing a foot switch. The scenario is based off [42], but was also referenced in other literature, such as [7]. For this process, the sequence of activities was modified compared to the referenced publications. In a first step, the human worker needs to prepare the left working area for the robot. Afterwards, the axis is moved to the left, and the left cobot can begin performing the left spot tape, while the human prepares the right working area. Once both steps are finished, the human moves the axis to the right, allowing the right cobot to work on the right spot tape, while the human needs to clean the left spot tape. Afterwards, the axis is moved to the right again , and the human worker starts cleaning the right spot tape, finishing the process. Due to the dependencies between the human worker and the robot, the "Cooperation" pattern is present in this process.

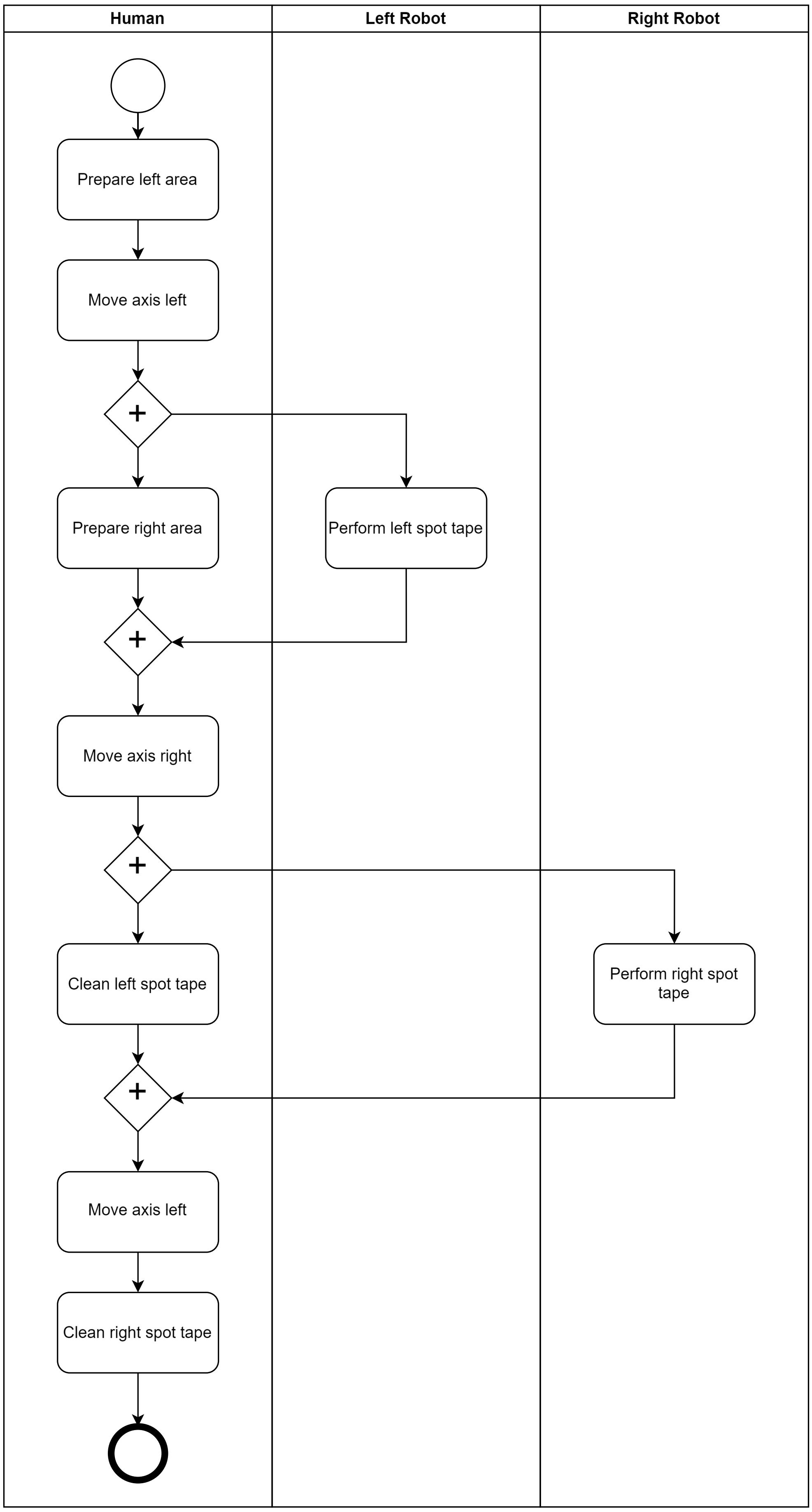


**Figure 5.4:** Spot Taping process

### 5.1.5 Cooperation: Refrigerator Assembly

[44] describes a process where refrigerators are assembled in a HRC scenario. The cobot has the task of applying insulation material to refrigerators, while the human workers checks the fridge for faulty spots and fixes them accordingly. While there is no time dependency between the two actors, there is a spacial constraint, as the human and robot cannot work on the same spot of the workpiece simultaneously. However, the two actors can work simultaneously at different areas of the workpiece, implying the "Cooperation" scenario.

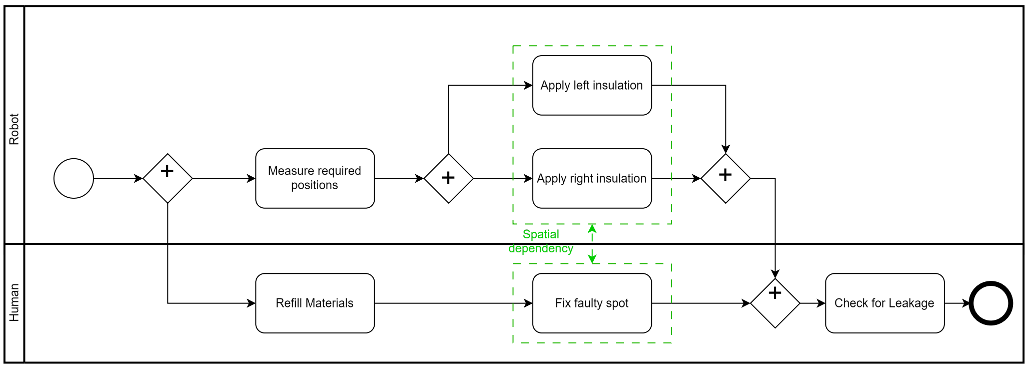


**Figure 5.5:** Refrigerator assembly process

### 5.1.6 Cooperation: Aircraft Cabin Part Assembly

Taken from [12], this use case describes the manufacturing of aircraft cabin parts. A human and a cobot work simultaneously on the same workpiece, however, they both have various tasks that do not have any timing constraints affecting each other. Still, both parties need to respect each others space, as they cannot work in the same spot. This spacial constraint once again entails the "Cooperation" scenario, as indicated by the marking in figure 5.6.

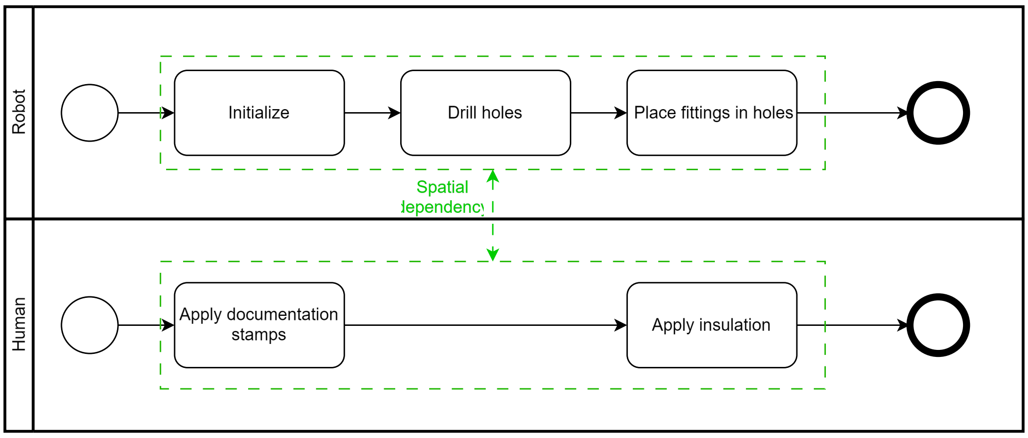


**Figure 5.6:** Aircraft cabin part assembly process

### 5.1.7 Collaboration: Box Assembly

In this scenario, taken from [36], the human and robot work together on building a wooden box. The robot helps the human by placing parts, but also by holding the existing workpiece in place so that the human can fit additional parts to the product. In 5.7, those supporting tasks by the robot are indicated with the orange-colored label "Supporting", while spacial constraints are again indicated by the green labels. The process has multiple steps, with this supportive case being present twice. Due to the existence of both spacial and temporal constraints between the human and the robot, this use case is of the "Collaboration" pattern. While this scenario may not be common in industrial settings, it helps illustrating joint work in HRC.

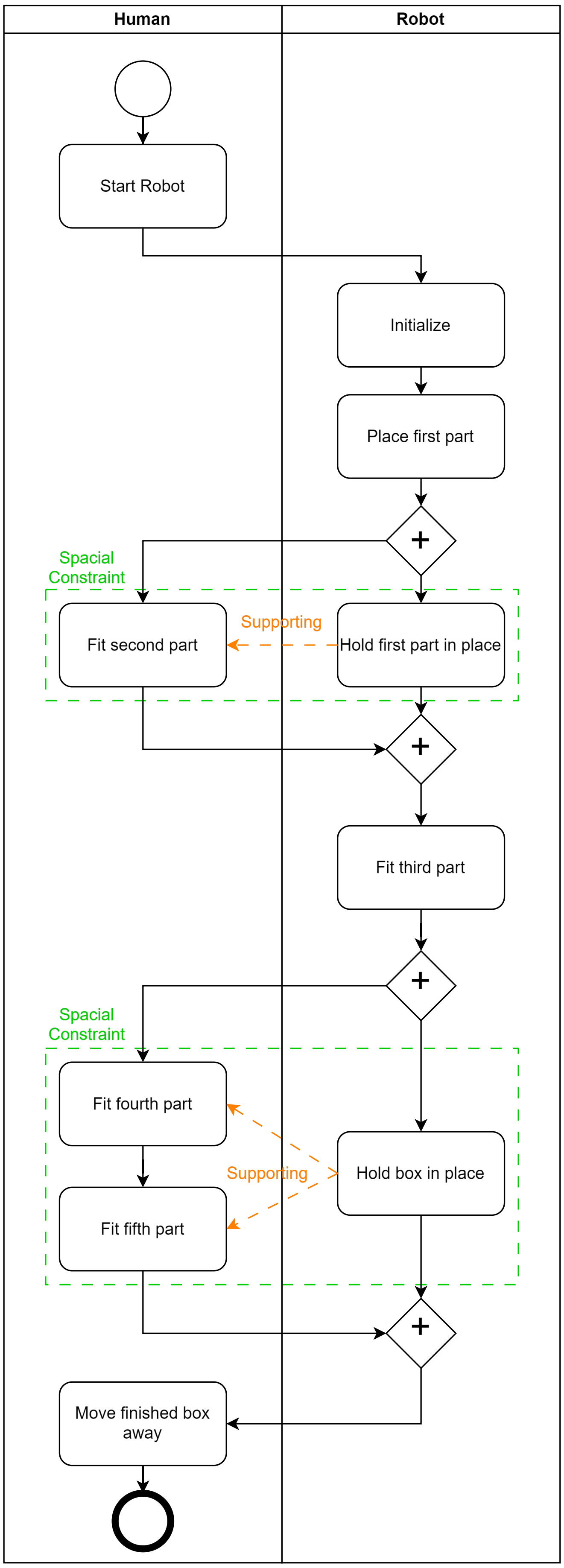


**Figure 5.7:** Box assembly process

### 5.1.8 Collaboration: Wheel Assembly

This use case is based on a production scenario in the automotive industry, which was described in [44]. In this situation, a high payload robot assists the human by handling the loading of the axles and rear wheel groups. The robot loads the axles independently. However, wheel assembly is a collaborative process where the robot manages the weight and the human manually adjusts the parts for precise positioning. The robot automatically picks up the wheel assemblies and positions them to the left and right of the axle, aligning them with the bolt holes. The human then manually guides the parts for accurate alignment with the holes.

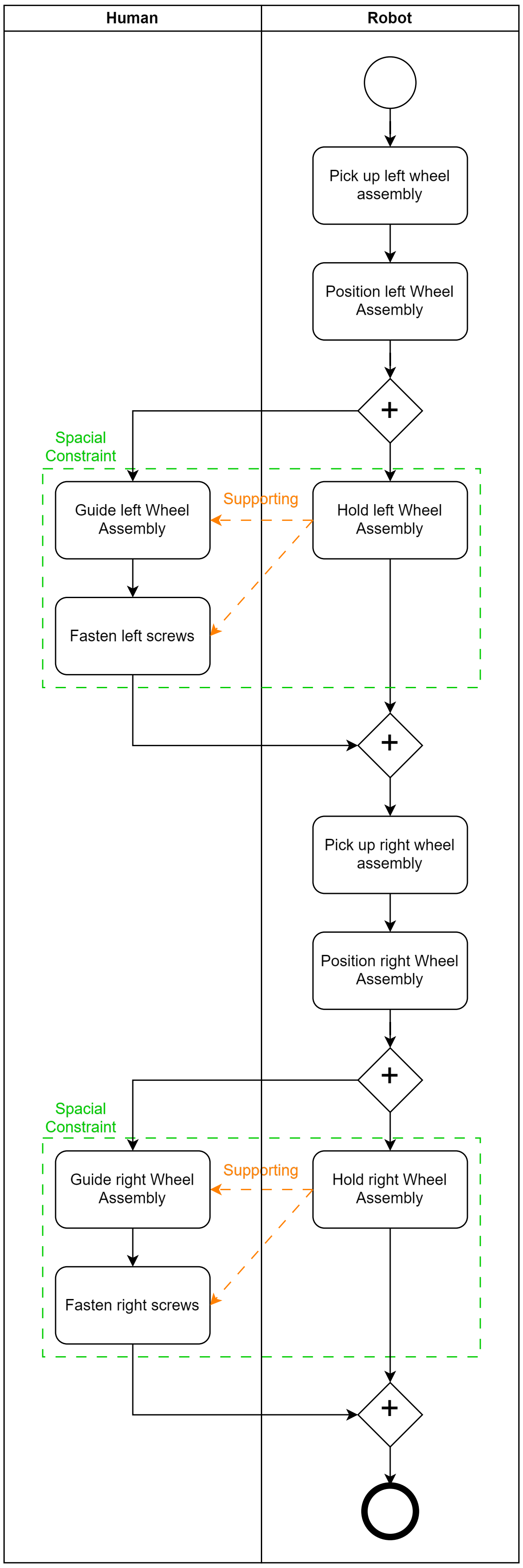

Human
Robot
Pick up left wheel assembly
Position left Wheel Assembly
Spacial Constraint
Guide left Wheel Assembly
Supporting
Hold left Wheel Assembly
Fasten left screws
Pick up right wheel assembly
Position right Wheel Assembly
Spacial Constraint
Guide right Wheel Assembly
Supporting
Hold right Wheel Assembly
Fasten right screws

**Figure 5.8:** Wheel assembly process

### 5.1.9 Internet data set: Processing and Quality Inspection of Components

In order to enable a unbiased evaluation of the algorithm with the help of practical data and not to rely solely on artificially generated data, a search was conducted on the internet for further data sets. The data set "Production Analysis with Process Mining Technology" [61] was found. The event log contains data from a production process, in particular cable head components. Due to the complexity and size of the process model, a process model is not presented here.

## 5.2 Artificial Event Log Generation

During the work on the algorithm, it became clear that in order to properly evaluate its accuracy, a larger number of event logs would be required. Thus, the decision was made to also develop and implement a tool to artificially generate event logs suitable for the algorithm. In chapter 9, the algorithm will be evaluated using both artificial event logs as well as event logs from online data sources. For the description of the implementation of the Event log generator, as well as a usage guide, please refer to chapter 8.

### 5.2.1 Generation of Event Logs for each Use Case

Using the event log generator described in 5.2, one event log was created for each of the use cases described in section 5.1. To strike a balance between practical relevance, low risk of bias and outliers, and reasonable computation time, 1000 cases was chosen as the length of the event log. A "delay" of up to 30 seconds has been configured between the activities. The duration of the activities of the individual use cases was chosen to be practice-oriented, with a suitable variance in each case. Table 5.2 shows an exert of the event log created for the "Refrigerator assembly" use case.

| case_id | activity | start_timestamp | end_timestamp | resource |
|---|---|---|---|---|
| 0 | Start Event | 09.07.2024 09:27:14 | 09.07.2024 09:27:14 | System |
| 0 | Measure required positions | 09.07.2024 09:27:34 | 09.07.2024 09:30:06 | Cobot |
| 0 | Refill materials | 09.07.2024 09:27:22 | 09.07.2024 09:28:34 | Operator |
| 0 | Fix faulty spot | 09.07.2024 09:30:16 | 09.07.2024 09:32:34 | Operator |
| 0 | Apply left insulation | 09.07.2024 09:30:24 | 09.07.2024 09:32:18 | Cobot |
| 0 | Apply right insulation | 09.07.2024 09:32:44 | 09.07.2024 09:35:20 | Cobot |
| 0 | Check for leakage | 09.07.2024 09:35:20 | 09.07.2024 09:37:27 | Operator |
| 0 | End Event | 09.07.2024 09:37:27 | 09.07.2024 09:37:27 | System |
| 1 | Start Event | 09.07.2024 09:27:14 | 09.07.2024 09:27:14 | System |
| 1 | Refill materials | 09.07.2024 09:27:42 | 09.07.2024 09:28:40 | Operator |

**Table 5.2:** Exemplary event log for the "Refrigerator assembly" use case

# Chapter 6

# Theoretical Development of the Algorithm

The following chapter will describe the fundamental ideas of the algorithm used to identify Collaborative patterns in an event log. General assumptions that need to be taken into consideration are laid out. Additionally, the target classification for a number of scenarios will be outlined.

## 6.1 General Assumptions

In order to develop a theoretical algorithm, a number of assumptions about the detection and logic must be made. These assumptions are listed below.

1. There is no singular workpiece ID in the event log, as one case within the event log (consisting of multiple log entries) always refers to the same workpiece (Case ID = Workpiece ID). During a case, other workpieces and supplies may be included in the finished product. However, these are not named; only the case ID is used to identify a case. For example, in a case, a component with workpiece ID 1 and another component with workpiece ID 2 can flow into a finished product with workpiece ID 3. In this case, all steps for the production of workpiece 3 have the case ID 3. The manufacturing of workpiece 1 however has the case ID 1, as long as it not manufactured entirely within case 3. The manufacturing of a large number of workpieces can still be counted as a singular case, as long as the workpieces do not have individual IDs within the case. As an example, one case may concern the manufacturing of 1000 screws, which are one unit, which is the case ID. An individual screw does not have a workpiece or case ID in this case.

2. Activities that count towards the "Collaboration" pattern may, but do not need to, have the same name. As an example, parallel activities "Hold Workpiece in Place" and "Drill Holes" may count towards the "Collaboration" pattern, but also if the parallel activities of both activities are called "Drill Holes".

3. Patterns should be classified for each activity, not for the whole case. However, activities that have a length of 0 (start-timestamp equals end-timestamp), such as system events, should not be classified as any collaborative pattern.
4. If an activity occurs several times under the same name within a case, it is counted as a single activity for the recognition of patterns. For example, if the activity "Drill holes" begins at time unit 1 and ends at time unit 5, and then occurs again from time unit 10-12, both occurrences have the same pattern of cooperation. This is the case, for example, in order to correctly classify breaks in processing such as the end of shifts.
5. If different activities within a case overlap in time, it is assumed that this work takes place at the same location.
6. Each log entry includes both the start timestamp as well as the end timestamp of the activity so that the temporal overlapping between activities can be observed.
7. Each log entry includes the resource performing it, which may be "Operator", "Cobot" or any other name.

Figure 6.1 lays out the relationship between the different attributes of the event logs used in this algorithmn.

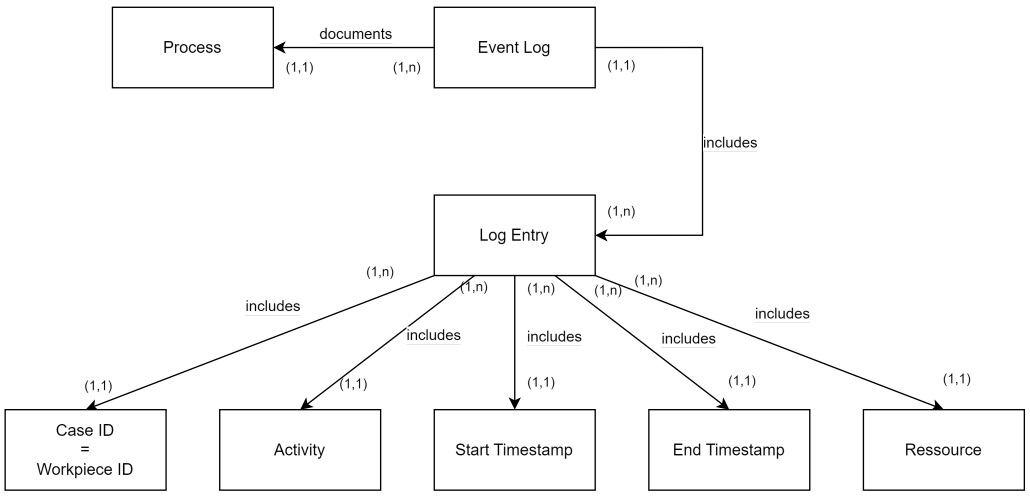


**Figure 6.1:** Metamodel defining the event logs utilized in the algorithm

Table 5.2 shows a possible event log that contains all required attributes for the algorithm.

## 6.2 Assumptions about the Classification of Patterns

The biggest challenge of the algorithm is the correct classification of collaboration patterns with limited information. This means that no qualitative analysis of the activities themselves is possible. Instead, activities are classified according to certain temporal overlaps based on theoretical assumptions. It should be noted again that, according to the

assumptions in 6.1, activities within a case always take place in the same place, provided they occur at the same time.
In this section, the logic of classification used by the algorithm will be laid out in detail. Each possible case of collaboration will be named and the respective classification will be described. Visualizations will be created using examples of two actors, however, all cases also apply to scenarios involving more than two actors.

### 6.2.1 Definition of the Threshold

In the algorithm, a important variable exists, which is called the "threshold". It is always set for each case, and is employed to identify different collaboration patterns between activities. Specifically, it helps to decide whether activities overlap significantly, thus deciding the classification of patterns, particularly whether "Cooperation" or "Collaboration" occurs. The threshold percentage defines the sensitivity of these pattern detections, influencing how strictly the overlap or timing relationships are evaluated. Analysis about the optimal length of the threshold value is described in section 9.2. In the following illustrations, the threshold will be visualized by a red box.

### 6.2.2 Coexistence Cases

As described in section 4.3.1, the "Coexistence" pattern is by definition not an actual collaborative scenario, as actors work on separate workpieces and/or processes, without interacting with each other. As soon as there are more than one actor present in one case, at least the pattern "Synchronized" is classified, meaning that the "Coexistence" pattern only occurs if a case is handled by a single actor alone.

**Coexistence case: One actor only**
In this base case, the whole case is performed by the same actor, which could be for example "Operator", "Cobot", or "CobotA". There are no other actors present in the case.

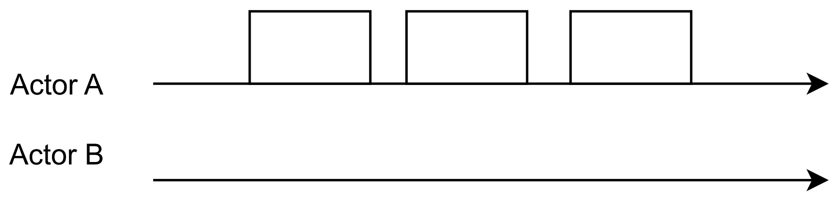


**Figure 6.2:** Coexistence case: "One actor only" (own illustration)

### 6.2.3 Synchronized Cases

The "Synchronized" Pattern implies the work of two or more actors within the same process, however, at different times.

**Synchronized case: No overlap**
In this case, no temporal overlap occurs at all, and activities start outside of the thresholds of neighboring activities.

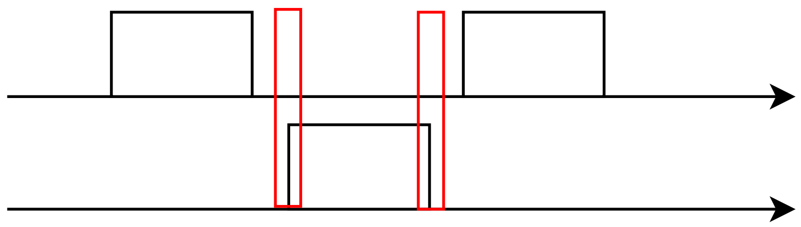


**Figure 6.3:** Synchronized case: "No overlap" (own illustration)

**Synchronized case: Barely no overlap**
Similar to the previous case, no temporal overlap occurs, however, activities start within the threshold value of their previous activity. As no overlap occurs, no "Cooperation" is implied, thus meaning the pattern "Synchronized" is present.

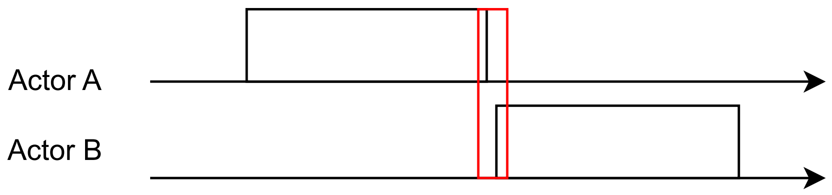


**Figure 6.4:** Synchronized case: "Barely no overlap" (own illustration)

### 6.2.4 Cooperation Cases

The pattern "Cooperation" is implied when work on one workpiece (within a case) occurs during the same time, however, at different positions on the workpiece. The actors do not actually work interactively, however, they need to be aware of each other and respect each others space. In the algorithm, differentiation between "Cooperation" and "Collaboration" is determined by classifying whether to simultaneous activities most likely have the same goal or not.

**Cooperation case: Overlap with other start/end**
An activity of actor B occurs during an activity of actor A, with a clearly different start and end. The different start and end indicate separate goals on the same workpiece.

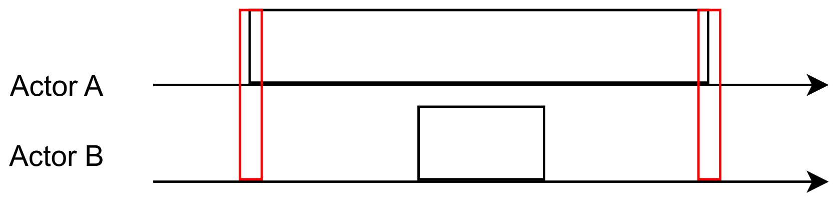


**Figure 6.5:** Cooperation case: "Overlap with other start/end" (own illustration)

**Cooperation case: Overlap at end**
An activity of actor B runs during an activity of actor A with a clearly different start but similar end. The different start indicates separate goals on the same workpiece.

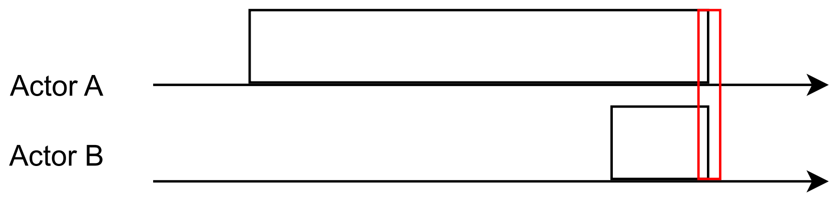


**Figure 6.6:** Cooperation case: "Overlap at end" (own illustration)

**Cooperation case: Overlap at beginning**
Similar to "Overlap at end", an activity of actor B runs during an activity of actor A, however, this time with a similar start and different end. The different end time once again indicates separate goals on the same workpiece.

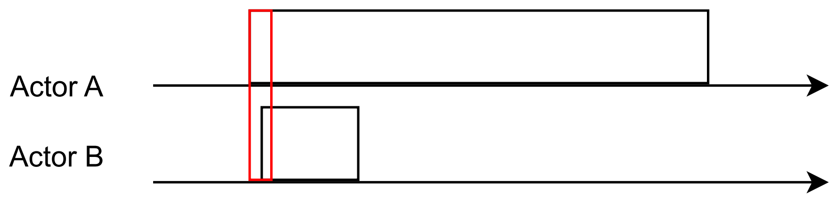


**Figure 6.7:** Cooperation case: "Overlap at beginning" (own illustration)

**Cooperation case: Multiple overlap at end**
Several activities of actor B happen during a single activity of actor A, but with a clearly different start. This different start indicates separate processes on the same workpiece.

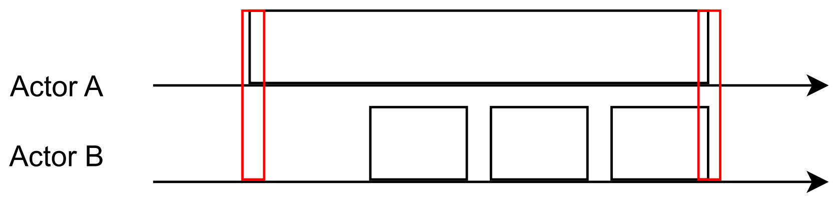


**Figure 6.8:** Cooperation case: "Multiple overlap at end" (own illustration)

**Cooperation case: Multiple overlap at beginning**
Similar to the previous case, multiple activities of actor B happen during a single activity of actor A, however, this time with a clearly different end. This different end indicates separate processes on the same workpiece.

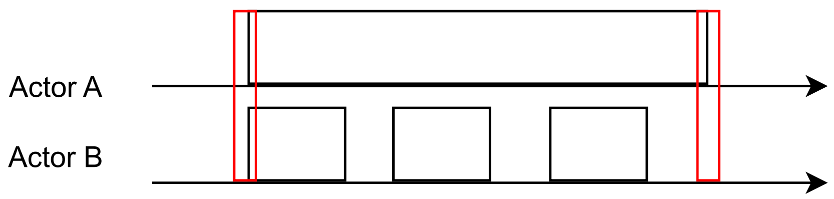


**Figure 6.9:** Cooperation case: "Multiple overlap at beginning" (own illustration)

**Cooperation case: Overlap in transition period**
An activity of actor B starts at the end of the first activity of actor A (within the threshold), and ends at the threshold of the start of the second activity of actor A. The short overlap with the activities of A indicates non-collaboration.

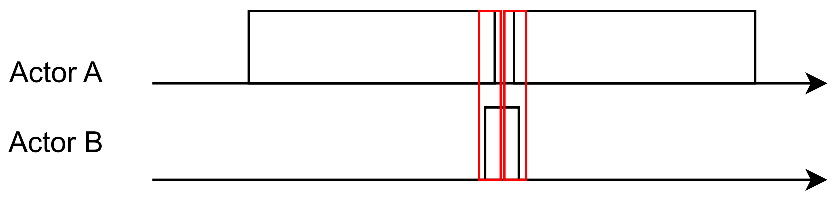


**Figure 6.10:** Cooperation case: "Overlap in transition period" (own illustration)

**Cooperation case: Partial overlap in transition period**
This case is similar to the previous one, however, the activity of Actor B which occurs in the transition period between two activities of actor A starts before and end after the respective threshold values. The earlier start and the later end of actor B speak against collaboration.

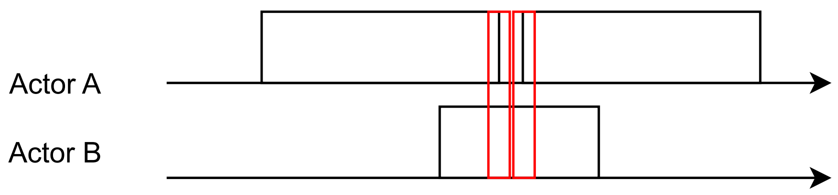


**Figure 6.11:** Cooperation case: "Partial overlap in transition period" (own illustration)

**Cooperation case: Overlap to start/end**
The activity of actor B begins once the activity of actor A has almost finished. This almost implies the "Synchronized" pattern, however, due to the temporal overlap, "Cooperation" was chosen.

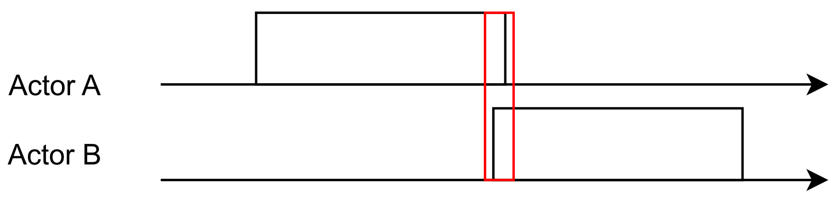


**Figure 6.12:** Cooperation case: "Overlap to start/end" (own illustration)

### 6.2.5 Collaboration Cases

The pattern "Collaboration" is implied when work on one workpiece (within a case) occurs during the same time, however, other than "Cooperation", all actors work interactively towards the same goal. In the algorithm, the "Collaboration" pattern was mainly detected using overlapping of activities.

**Collaboration case: Complete overlap**
Both actors begin and end their activities at the same time, which indicates a close interaction and dependence of their actions, which is why "collaboration" is most likely to apply.

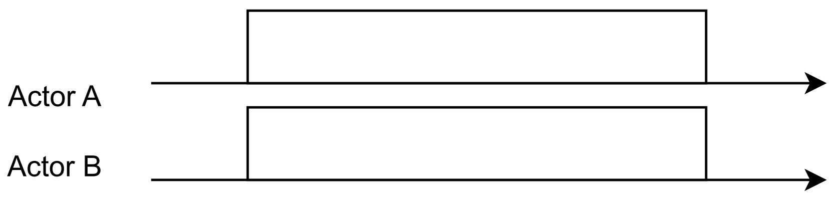


**Figure 6.13:** Collaboration case: "Complete overlap" (own illustration)

**Collaboration case: Nearly complete overlap**
All actors begin and end their activities within the threshold of each other, which once again indicates a close interaction and dependence of their actions.

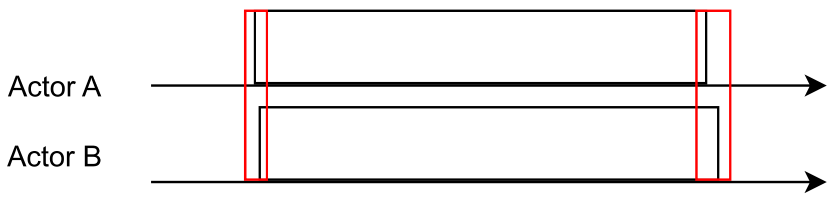


**Figure 6.14:** Collaboration case: "Nearly complete overlap" (own illustration)

**Collaboration case: Multiple overlapping**
During one longer activity of actor B, multiple (2 or more) shorter activities of actor B occur. The simultaneous start and end indicates interrelated processes on the same workpiece. It is assumed that the activities of both actors actively complement each other,

which implies "Collaboration". In cases involving more than two actors, please note that the activities of the third actor C will not be counted towards the collaboration group between actors A and B unless they also start around the start and end thresholds of actor A's activity.

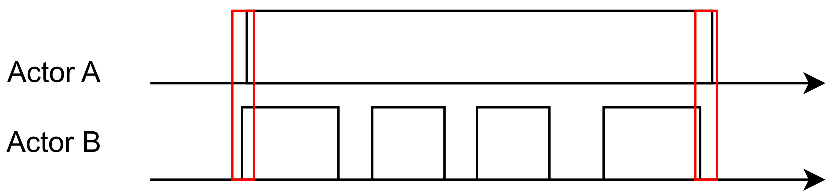


**Figure 6.15:** Collaboration case: "Multiple overlapping" (own illustration)

## 6.3 Logic of the Algorithm

In the following section, the sequence of functions in the algorithm will be described.

### 6.3.1 High-level Sequence

Figure 6.16 shows a high-level flow-chart depicting the logic of the algorithm. In the red boxes, the cases defined in 6.1 are placed in the flowchart, next to the function that is used to detect them. Generally, there are six checks for patterns present, which are performed in this order:

- **Get number of actors**: This simple function only checks whether the number of actors equals "one" for a case, implying "Coexistence".
- **Check for Collaboration Group**: This function is only used to detect the "Multiple Overlapping" case.
- **Check for Collaboration**: This function discovers the remaining "Collaboration" cases, which are "Complete overlap" and "Nearly complete overlap".
- **Check for Cooperation**: This function is used to discover all possible "Cooperation" cases.
- **Check for Synchronized**: Cases that do not fall under the above ones are classified by this function.

The patterns present are not identified on a per-case base, but for each combination of activities within a case. If a combination of activities falls within more than one pattern, the highest ranking pattern will be applied. The ranking of patterns is laid out in table 6.1. The pattern "Coexistence" does not need a ranking, as it always applies to either all or none of the activities in a case (see case "One Actor only" in section 6.1).

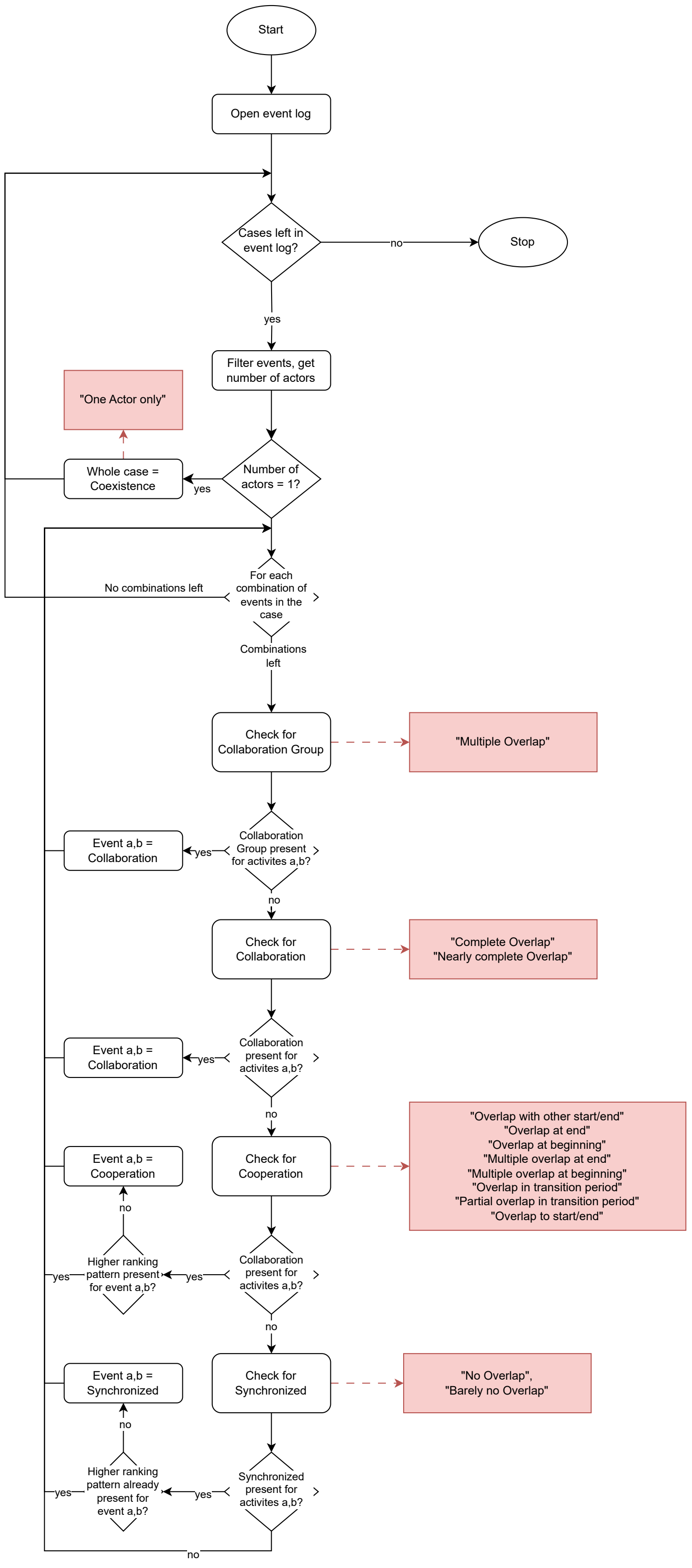

Start
Open event log
Cases left in event log?
no
Stop
yes
Filter events, get number of actors
"One Actor only"
Whole case = Coexistence
yes
Number of actors = 1?
No combinations left
For each combination of events in the case
Combinations left
Check for Collaboration Group
"Multiple Overlap"
Event a,b = Collaboration
yes
Collaboration Group present for activites a,b?
no
Check for Collaboration
"Complete Overlap" "Nearly complete Overlap"
Event a,b = Collaboration
yes
Collaboration present for activites a,b?
no
Event a,b = Cooperation
Check for Cooperation
"Overlap with other start/end" "Overlap at end" "Overlap at beginning" "Multiple overlap at end" "Multiple overlap at beginning" "Overlap in transition period" "Partial overlap in transition period" "Overlap to start/end"
no
Higher ranking pattern present for event a,b?
yes
yes
Collaboration present for activites a,b?
no
Event a,b = Synchronized
Check for Synchronized
"No Overlap", "Barely no Overlap"
no
Higher ranking pattern already present for event a,b?
yes
yes
Synchronized present for activites a,b?
no

**Figure 6.16:** Flowchart depicting the high-level sequence of the algorithm

| Collaborative Pattern | Rank |
|---|---|
| Collaboration | 1 |
| Cooperation | 2 |
| Synchronized | 3 |
| Coexistence | No Ranking |

**Table 6.1:** Ranking between Collaborative Patterns in the algorithm

The functions in this list will be described in more details in subsection 6.3.2.

### 6.3.2 Checks for Individual Patterns

In this function, the functions for identifying individual patterns between two activities a,b are detailed. Pseudo code shows the exact logic for each function. Please note that in the pseudo code, one object such as "a" contains all information about one event of a case, such as this example:

```
0,Refill materials,2024-07-09 09:27:22,2024-07-09 09:28:34,Operator
```

**Listing 6.1:** Exemplary event object

For simplicity, in the pseudo code the program can access fields of the data object using *a.attribute*, such as *a.Resource*, which would return "*Operator*" in this example.

#### General Sequence

The function find_patterns_in_case is used to call the other functions used to identify individual patterns. Is is called for each case found in the event log. After creating data objects (lists) and calculating the case-value for the threshold, it is checked whether the number of actors exceeds one. If this is not the case, the pattern "Coexistence" is applied to all events of the case. Otherwise, the functions to check for collaboration groups, collaboration, cooperation and synchronized cases are called (in this order). If a combination of activities has more than one applicable, the highest ranking pattern will be counted, as described in section 6.2 and table 6.1. The logic to do this is not represented in the following pseudo code, and in an actual implementation a function would check for this if a pattern is updated (see lines 18, 20, 22 and 24 in the listing).

```
1  FUNCTION find_patterns_in_case(case_id, case_events, threshold_percentage)
2
3      Set Threshold according to user-input (threshold_percentage)
4
5      Get number of actors
6
7      IF number of actors is 1 THEN
8          FOR event IN case_events DO
9              Update Pattern to coexistence
10     ELSE
11         FOR i FROM 0 TO length(case_events) - 1 DO
12             FOR j FROM i + 1 TO length(case_events) - 1 DO
13
14                 a = case_events[i]
```

```
            b = case_events[j]

            IF is_collaboration_group(case_events, a, b, threshold) THEN
                Update pattern of a and b to "Collaboration"
            ELSE IF is_collaboration(a, b, threshold) THEN
                Update pattern of a and b to "Collaboration"
            ELSE IF is_cooperation(a, b, threshold) THEN
                Update pattern of a and b to "Cooperation"
            ELSE IF is_synchronized(a, b, threshold) THEN
                Update pattern of a and b to "Synchronized"
```

**Listing 6.2:** Pseudocode for finding patterns in cases

### Check for Collaboration Group

This function is required to check for the "Multiple Overlap" case. What is unique about this function compared to other checks is that the parameters of the function include all events belonging to the case, not just the two activities that are checked at a time.
In summary, the first step is to check whether the two activities being checked overlap and whether the resources are different. For reasons of simplicity, if a's duration is less than b's duration, the two data objects will be swapped. In addition, a check is carried out to ensure that activity b does not end later than the threshold phase after the end of activity a.
If the conditions are met, a list of shorter activities that use a different resource than a and that take place during the longer activity a is created.
If the list is not empty, the system checks whether the first shorter activity on the resource of activity b starts near the start of activity a and the last shorter activity ends near the end of activity a.
The following pseudo code shows the logic of the "Check for Collaboration Group"function, which will be called "is_collaboration_group" in the following.

```
FUNCTION is_collaboration_group(case_events, a, b, threshold)

    Calculate overlap_start and overlap_end

    IF a.resource is equal to b.resource OR overlap_start is greater than
        overlap_end THEN
        RETURN False

    IF a.end_timestamp + threshold is less than or equal to b.end_timestamp THEN
        RETURN False

    FOR each event IN case_events DO
        IF event.resource. is not equal to a.resource THEN
            IF event's duration falls within a's time window and threshold THEN
                IF event's duration is less than or equal to a's duration THEN
                    ADD event TO shorter_activities

    IF shorter_activities is not empty THEN

        Get first and last shorter acitvity

        IF first_short_activity.start_timestamp is less than or equal to a.
            start_timestamp + threshold AND
            last_short_activity.end_timestamp is greater than or equal to a.
                end_timestamp - threshold THEN
                RETURN True

    RETURN False
```

**Listing 6.3:** Pseudocode for checking for Collaboration Groups

### Check for Collaboration

This function checks for the remaining "Collaboration" Cases, which are "Complete overlap" and "Nearly complete overlap". The function first checks whether the activities a and b use a different resource. Afterwards, it checks for a overlap within the threshold periods near the start and end of the activities.

```
FUNCTION is_collaboration(a, b, threshold)
    IF a.resource is equal to b.resource THEN
        RETURN False

    IF absolute difference between a.start_timestamp and b.start_timestamp is
        less than or equal to threshold AND
        absolute difference between a.end_timestamp and b.end_timestamp is less
            than or equal to threshold THEN
            RETURN True

    RETURN False
```

**Listing 6.4:** Pseudocode for checking for Collaboration

### Check for Cooperation

Similar to the "is_collaboration" function previously described, the is_cooperation function checks whether activities a,b utilize different resources and have any temporal overlap. The key difference is that "Cooperation" is assigned once there is any overlap.

```
FUNCTION is_cooperation(a, b, threshold)
    IF a.resource is equal to b.resource THEN
        RETURN False

    IF (a.start_timestamp < b.start_timestamp < a.end_timestamp OR
        a.start_timestamp < b.end_timestamp < a.end_timestamp OR
        b.start_timestamp < a.start_timestamp < b.end_timestamp OR
        b.start_timestamp < a.end_timestamp < b.end_timestamp) THEN
            RETURN True

    RETURN False
```

**Listing 6.5:** Pseudocode for checking for Cooperation

### Check for Synchronized

If all other cases have not been met, the "is_synchronized" function is used to declare pairs of activities as "Synchronized". The only check present is if the activities do not overlap, including the threshold before and after each activity.

```
FUNCTION is_synchronized(a, b, threshold)
    IF b.start_timestamp > a.end_timestamp + threshold OR
       a.start_timestamp > b.end_timestamp + threshold THEN
        RETURN True

    RETURN False
```

**Listing 6.6:** Pseudocode for checking for Synchronized

# Chapter 7

# Implementation of the Algorithm

## 7.1 Requirements to the Implementation

In order to meet a high standard, the implementation of the algorithm must meet a number of requirements, which are listed in the following section.

- **Ease of use**: The use of the program must be intuitive and largely self-explanatory. The use of a graphical user interface is advantageous for this.
- **Comprehensive options for analysis**: The program should offer options for precisely tracking classifications of individual activities and cases.
- **High performance**: The program must be able to classify Collaborative Patterns (depending on the length of the event log) in a reasonable amount of time.
- **Graphical representation of cases**: The program should be able to display individual cases and their classification in graphical form.
- **Process discovery**: For additional clarity, the program should be able to create and display process models of the event log.
- **High compatibility**: The program must be able to run on all common operating systems.

## 7.2 Setup Guide

In this section, the steps required to run the script are described in detail. Additionally, the utilized software and libraries are discussed.

### 7.2.1 Required Software

The only software required to run the script is a recent version of Python, such as Python 3.12.2. Other installations are not required, however, in order to manipulate .csv Event logs, spreadsheet software such as Microsoft Excel or Notepad-software such as Notepad++ may be useful.

### 7.2.2 Required Python-Libraries

This subsection details the libraries that must be installed before running the script.

**Pandas:** The Pandas library is a powerful data manipulation and analysis tool. In this software, it is used to read and process event log data from CSV files. Pandas is used to create data frames from the event logs, which simplifies data manipulation tasks such as filtering, grouping, and calculating statistics. It also helps to convert timestamp strings into datetime objects for easier time-based calculations. The average duration of activities, which is essential for identifying patterns, is calculated using pandas.

**Pm4py:** The Pm4py (Process Mining for Python) library [62] is designed for Process Mining tasks. It provides tools for the discovery, visualization, and analysis of process models from event logs. In this script, Pm4py is used to:

- Discover process models from event logs using the inductive miner
- Visualize Petri nets, including decorating them with additional information like activity labels and resource involvement.

**Plotly:** The Plotly library is a graphing library that can create interactive, high-quality graphs. This script uses Plotly.express to create interactive Gantt charts. These charts help visualize the timeline of activities within each case from the event log. The graphs are enhanced with annotations to indicate detected patterns (such as "Collaboration" or "Cooperation"), making it easier to understand the process flow and relationships between activities.

**Pillow:** The Pillow library is used to open and display various image file formats. In this script, Pillow is used specifically to handle image data for displaying the Petri net visualizations within the Graphical User Interface (GUI). Pillow's ImageTk module helps to convert images into a format suitable for displaying in a Tkinter canvas widget, making it possible to visually represent the process models discovered by the script.

**Customtkinter:** The Customtkinter library is an extension of the standard Tkinter library that provides custom widgets and a modern look and feel for GUI applications. This script uses Customtkinter to create a user-friendly GUI that allows users to interact with the event log, configure thresholds, view detected patterns, and visualize process models. The library helps create various GUI components such as buttons, labels, frames, and more, ensuring a consistent and visually pleasing interface.

### 7.2.3 Running the Script

Assuming Python is installed on the PC used to run the script, the installation of the aforementioned libraries is necessary. This is achieved using the following Terminal-

commands:
**Windows:**

```
pip install Pillow pandas pm4py plotly customtkinter tk
```

**Linux (Debian-based):**

```
sudo apt install python3-tk
pip3 install Pillow pandas pm4py plotly customtkinter tk
```

**MacOS:**

```
xcode-select --install
pip3 install Pillow pandas pm4py plotly customtkinter tk
brew install python-tk
```

After the required libraries have been installed, the following commands can be used to run the program:
**Windows:**

```
cd path\to\script
python CollaborativePatternIdentification.py
```

**Linux (Debian-based):**

```
cd path\to\script
python3 CollaborativePatternIdentification.py
```

**MacOS:**

```
cd path\to\script
python3 CollaborativePatternIdentification.py
```

After opening the program from the command line, all remaining steps are performed in the GUI. No further command line interaction is required or possible, although debugging information about detected patterns is printed to the command line.

## 7.3 Requirements to the Event Log

Due to the way the program was implemented, event logs that are to be analyzed with the program need to follow a specific naming scheme for their relevant attributes. This only concerns the attributes described in section 6.1 and figure 6.1. Any other attributes of the event log are not taken into consideration. The attributes must be named as follows:

| Attribute | Attribute name in the program |
|---|---|
| Case identifier | case_id |
| Activity identifier | activity |
| Start timestamp | start_timestamp |
| End timestamp | end_timestamp |
| Resource | resource |

**Table 7.1:** Naming scheme for attributes in the event log

## 7.4 Implementation in Python of the Classification Algorithm

In this section, the implementation of the Classification software will be described and documented in detail. Each section and function of the program will be included and described. The purpose of this section is so that readers can understand the exact methods of working of the implementation and make changes according to their requirements if necessary. Please note that the order in which program functions are explained in this section is not necessarily the order in which they are called during the runtime of the program.
The first listing, 7.1 contains the import of all required libraries. Additionally, a dictionary is created, which contains the ranking for all Collaboration patterns. This is later required to decide which pattern can replace another pattern if both have been detected for one activity.

```python
import pm4py
from pm4py.visualization.petri_net import visualizer as pn_visualizer
from pm4py.visualization.petri_net.common import visualize as pn_visualize
import plotly.express as px
import random
import threading

# Dictionary containing the ranking between patterns:
PATTERN_RANK = {
    'Collaboration': 1,
    'Cooperation': 2,
    'Synchronized': 3,
    'Coexistence': 4
}
```

**Listing 7.1:** Classification Algorithm: Import of libraries, pattern ranking

The function "read_event_log" is called in order create an internal list of all events from the CSV file which was read as an event log. The purpose of this is to greatly speed up the processing, as no file operations on the disk of the computer are necessary afterwards. Also, further operations such as sorting the list are possible.

```python
def read_event_log(file_path):
    # Create list of events from event log (from CSV file)
    events = []
    with open(file_path, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            events.append({
                'case_id': row['case_id'],
                'activity': row['activity'],
                'start_timestamp': pd.to_datetime(row['start_timestamp']),
                'end_timestamp': pd.to_datetime(row['end_timestamp']) if row['
                    end_timestamp'] else None,
                'resource': row['resource']
            })
    return events
```

**Listing 7.2:** Classification Algorithm: read_event_log

The function "update_pattern" is called every time the algorithm assigns a pattern to an activity. The function then checks, according to the ranking of the pattern (see listing 7.1), if the newly assigned pattern has a higher rank than the existing pattern. If the rank of the new pattern is higher (lower numerical value) than that of the current one, the function updates the pattern for this activity in the pattern dictionary and resets the list of contributing activities so that it only contains the contributing activity. If the rank of the new pattern is the same as that of the current pattern, the contributing activity is added to the list of contributing activities for this pattern. If no pattern exists yet, the new pattern will be assigned and the contributing activity will be first entry on the list.

```
def read_event_log(file_path):
    # Create list of events from event log (from CSV file)
    events = []
    with open(file_path, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            events.append({
                'case_id': row['case_id'],
                'activity': row['activity'],
                'start_timestamp': pd.to_datetime(row['start_timestamp']),
                'end_timestamp': pd.to_datetime(row['end_timestamp']) if row['
                    end_timestamp'] else None,
                'resource': row['resource']
            })
    return events
```

**Listing 7.3:** Classification Algorithm: update_pattern

Function "find_patterns_in_case", listing 7.4, is called for each case that was found in the event log. It receives the relevant case id and a list of all events of that case, alongside the threshold value. In this implementation, the default value for the threshold is 30 % of the mean duration of activities, however, if the user has set a different value in the GUI, this value is used instead. In the first part, the function creates empty lists for the patterns and contributing activities. Additionally, system events are filtered out and the exact threshold value for this case is calculated. Afterwards, the checks for individual patterns are performed, starting with the check for "Coexistence". This is the case if the number of actors is exactly one. Otherwise, the checks for Collaboration Group, Collaboration, Cooperation and Synchronized are performed one after the other. After the algorithm iterated over each combination of activities within the case, the list of patterns, contributing activities and the threshold value are returned for the case.

```
def find_patterns_in_case(case_id, case_events, threshold_percentage=30):
    patterns = {}
    # Create a list of contributing activities for each activity
    contributing_activities = defaultdict(list)
    # Filter out events that don't have an end timestamp or have the same start
        and end timestamp (System events)
    case_events = [event for event in case_events if event['end_timestamp'] and
        event['start_timestamp'] != event['end_timestamp']]
    # Calculate the mean length of activities in the case
    mean_duration = pd.Series([(event['end_timestamp'] - event['start_timestamp
        ']).total_seconds() for event in case_events]).mean()
    threshold = pd.Timedelta(seconds=(mean_duration * threshold_percentage /
        100))
    # Get a list of unique resources (actors) from the case events
    actors = list(set(event['resource'] for event in case_events))

    if len(actors) == 1:
        # Base case: If only one actor present (System events do not count),
            pattern is Coexistence for every activity of this case.
        for event in case_events:
            update_pattern(patterns, event['activity'], 'Coexistence',
                contributing_activities, event['activity'])
    else:
        # For each event of the case:
        for i in range(len(case_events)):
            for j in range(i + 1, len(case_events)):
                # For each combination of events in the case:
                a, b = case_events[i], case_events[j]
                # Check for Collaboration group (1:n) (a = long activity, b= one
                     of the short activities)
                if is_collaboration_group(case_events, a, b, threshold):
                    print(f"Case {case_id}: Collaboration Group detected between
                         {a['activity']} and {b['activity']}")
                    update_pattern(patterns, a['activity'], 'Collaboration',
                        contributing_activities, b['activity'])
                    update_pattern(patterns, b['activity'], 'Collaboration',
                        contributing_activities, a['activity'])
                # Check for collaboration (1:1)
                elif is_collaboration(a, b, threshold):
                    print(f"Case {case_id}: Collaboration detected between {a['
                        activity']} and {b['activity']}")
                    update_pattern(patterns, a['activity'], 'Collaboration',
                        contributing_activities, b['activity'])
                    update_pattern(patterns, b['activity'], 'Collaboration',
                        contributing_activities, a['activity'])
                # Check for Cooperation (1:1)
                elif is_cooperation(a, b, threshold):
                    print(f"Case {case_id}: Cooperation detected between {a['
                        activity']} and {b['activity']}")
                    update_pattern(patterns, a['activity'], 'Cooperation',
                        contributing_activities, b['activity'])
                    update_pattern(patterns, b['activity'], 'Cooperation',
                        contributing_activities, a['activity'])
                # Check for Synchronized (1:1)
                elif is_synchronized(a, b, threshold):
                    print(f"Case {case_id}: Synchronized detected between {a['
                        activity']} and {b['activity']}")
                    update_pattern(patterns, a['activity'], 'Synchronized',
                        contributing_activities, b['activity'])
                    update_pattern(patterns, b['activity'], 'Synchronized',
                        contributing_activities, a['activity'])

    return case_id, patterns, contributing_activities, threshold
```

**Listing 7.4:** Classification Algorithm: find_patterns_in_case

Listing 7.5 contains the functions used to identify the patterns "Synchronized" (is_synchronized), "Collaboration" (is_collaboration) and "Cooperation" (is_cooperation). "Is_synchronized" checks for synchronization by ensuring that one activity starts after the other ends, including the threshold value. "Is_collaboration" checks if the overlap between activities occurs within the threshold values. "Is_cooperation" is similar, however, it returns "True" if any overlap occurs between the activities, even when outside the threshold value.

```
def is_synchronized(a, b, threshold):
    # If no overlap or short overlap occurs between events, even when they are
        performed by the same actor.
    # Function checks if event a starts only after event b has ended and there
        is no overlap, including threshold.
    if b['start_timestamp'] > a['end_timestamp'] + threshold or a['
        start_timestamp'] > b['end_timestamp'] + threshold:
        return True
    return False

def is_collaboration(a, b, threshold):
    # If resources are the same, no collaboration occurs between them.
    if a['resource'] == b['resource']:
        return False
    # If overlap occurs (also within threshold), pattern = collaboration.
    if abs(a['start_timestamp'] - b['start_timestamp']) <= threshold and abs(a['
        end_timestamp'] - b['end_timestamp']) <= threshold:
        return True
    return False

def is_cooperation(a, b, threshold):
    # Check if resources are different
    if a['resource'] == b['resource']:
        return False
    # Check if Activities overlap.
    if (a['start_timestamp'] < b['start_timestamp'] < a['end_timestamp'] or
        a['start_timestamp'] < b['end_timestamp'] < a['end_timestamp'] or
        b['start_timestamp'] < a['start_timestamp'] < b['end_timestamp'] or
        b['start_timestamp'] < a['end_timestamp'] < b['end_timestamp']):
        return True
    return False
```

**Listing 7.5:** Classification Algorithm: is_synchronized, is_collaboration, is_cooperation

The function "is_collaboration_group" is used exclusively to detect the "Multiple Overlapping" case described in 6.2.5. Other than the other three functions shown in Listing 7.5, it also receives a list of all events (=activities) of the current case, alongside the two activities to be checked (a and b), as well as the threshold value. It first checks if the activities are of different resources and actually overlap. It then swaps the data objects of a and b if b has a longer processing time. This is done to avoid implementing a similar functionality twice. Afterwards, an additional check is performed if the end timestamp of b is not after the end timestamp of a plus the threshold. This is necessary because otherwise "Collaboration" would be classified if b starts during a and ends much later and there are other activities c (,d...) that take place during a and have a similar end timestamp. If this is not the case, a list of shorter activities is then created, and all activities that occur during a (including threshold) and have a shorter duration than a are added to the list. If the list is not empty, the first and last of those shorter activities are determined. It is no problem if both the first and last shorter activity are the same activity, as this would be the "Nearly Complete Overlap" case, also described in 6.2.5. At the end of the function, it is checked if the first shorter activity starts near the start of a, and the last shorter activity ends near the end of a. If both requirements are met, "True" is returned.

```
def is_collaboration_group(case_events, a, b, threshold):
    overlap_start = max(a['start_timestamp'], b['start_timestamp'])
    overlap_end = min(a['end_timestamp'], b['end_timestamp'])

    # Check if activities actually overlap and if resources are different
    if a['resource'] == b['resource'] or overlap_start > overlap_end:
        return False
    # If b is the longer event, swap a and b. (simplifies rest of this function)
    if a['end_timestamp'] - a['start_timestamp'] < b['end_timestamp'] - b['
        start_timestamp']:
        a, b = b, a

    # If the end timestamp of b is later than the end timestamp of a + threshold
        , no collaboration occurs.
    # Necessary because otherwise collaboration would be classified if b begins
        during a and ends much later, and there are other activities c(,d...)
        that occur during a and have a similar end timestamp.
    if a['end_timestamp'] + threshold <= b['end_timestamp']:
        return False

    # Collect shorter activities of b's resource during and around a's duration
    shorter_activities = []
    for event in case_events:
        if event['resource'] != a['resource']:
            # Check if the event falls within the time window of a or the
                thresholds
            if (event['start_timestamp'] >= a['start_timestamp'] - threshold and
                event['end_timestamp'] <= a['end_timestamp'] + threshold):
                # Check if the duration of the event is shorter than the
                    duration of a
                if (event['end_timestamp'] - event['start_timestamp']) <= (a['
                    end_timestamp'] - a['start_timestamp']):
                    shorter_activities.append(event)

    # Check if the shorter activities exist:
    if shorter_activities:
        first_short_activity = min(shorter_activities, key=lambda x: x['
            start_timestamp'])
        last_short_activity = max(shorter_activities, key=lambda x: x['
            end_timestamp'])


        # Ensure the shorter activities are fully within the threshold period
        if (first_short_activity['start_timestamp'] <= (a['start_timestamp'] +
            threshold) and
            last_short_activity['end_timestamp'] >= (a['end_timestamp'] -
                threshold)):
            return True

    return False
```

**Listing 7.6:** Classification Algorithm: is_collaboration_group

The following functions are all included in the "CollaborationPatternApp" class, which encapsulates the GUI and its related functionality for the application. The "__init__" function, which is shown in listing 7.7 is called as soon as the program is launched, and is used to create all necessary elements of the GUI. Some elements, e.g. the "self.case_list_frame" (List of cases shown after opening an event log) are initially hidden in the program.

```
class CollaborationPatternApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Collaboration Pattern Classifier")

        self.frame = ctk.CTkFrame(root)
        self.frame.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)

        # Make the main frame expand with the window
        root.columnconfigure(0, weight=1)
        root.rowconfigure(0, weight=1)
        self.frame.columnconfigure(0, weight=1)
        self.frame.rowconfigure(4, weight=1)
        self.frame.rowconfigure(5, weight=1)
        self.frame.rowconfigure(7, weight=1)

        self.title_label = ctk.CTkLabel(self.frame, text="Collaborative Pattern
            Identification", font=("Arial", 16, "bold"))
        self.title_label.grid(row=0, column=0, padx=5, pady=5, sticky="w")

        self.help_button = ctk.CTkButton(self.frame, text="?", width=30, command
            =self.show_help)
        self.help_button.grid(row=0, column=2, padx=5, pady=5, sticky="e")

        self.threshold_label = ctk.CTkLabel(self.frame, text="Threshold (% of
            mean duration):", font=("Arial", 12))
        self.threshold_label.grid(row=1, column=0, padx=5, pady=5, sticky="w")
        self.threshold_label_ttp = self.create_tooltip(self.threshold_label, "
            Defines the threshold percentage of the mean activity duration used
             to determine Collaboration.")

        self.threshold_entry = ctk.CTkEntry(self.frame, font=("Arial", 12))
        self.threshold_entry.grid(row=2, column=0, padx=5, pady=5, sticky="w")
        self.threshold_entry_ttp = self.create_tooltip(self.threshold_entry, "
            Defines the threshold percentage of the mean activity duration used
             to determine Collaboration.")
        self.threshold_entry.insert(0, "30")  # Default value of 30%

        self.load_button = ctk.CTkButton(self.frame, text="Load Event Log",
            command=self.load_event_log, font=("Arial", 12))
        self.load_button.grid(row=3, column=0, padx=5, pady=5, sticky="w")
        self.load_button_ttp = self.create_tooltip(self.load_button, "Imports a
            .csv event log and discovers Collaborative patterns. Please set the
             Threshold value accordingly before loading the event log.")

        self.view_model_button = ctk.CTkButton(self.frame, text="View Process
            Model", command=self.display_petri_net, font=("Arial", 12))
        self.view_model_button.grid(row=4, column=0, padx=(5, 0), pady=5, sticky
            ="w")
        self.view_model_button.grid_remove()  # Initially hidden
        self.view_model_button_ttp = self.create_tooltip(self.view_model_button, "Discovers and displays a process model using the Inductive Miner")

        self.analyze_variants_button = ctk.CTkButton(self.frame, text="Analyze Process Variants", command=self.analyze_process_variants, font=("Arial", 12))
        self.analyze_variants_button.grid(row=4, column=1, padx=(5, 0), pady=5, sticky="w")
        self.analyze_variants_button.grid_remove()  # Initially hidden
        self.analyze_variants_button_ttp = self.create_tooltip(self.analyze_variants_button, "Shows an overview over the existing process variants and which patterns were discovered in them")

        self.save_button = ctk.CTkButton(self.frame, text="Save as CSV", command=self.save_as_csv, font=("Arial", 12))
        self.save_button.grid(row=4, column=2, padx=(5, 0), pady=5, sticky="w")
        self.save_button.grid_remove()  # Initially hidden
        self.save_button_ttp = self.create_tooltip(self.save_button, "Saves the current list of cases and patterns as a CSV file")

        self.case_list_frame = ctk.CTkFrame(self.frame)
        self.case_list_frame.grid(row=5, column=0, columnspan=3, padx=5, pady=5, sticky="nsew")
        self.case_list_frame.grid_remove()  # Initially hidden

        # Make the case_list_frame expandable
        self.case_list_frame.columnconfigure(0, weight=1)
        self.case_list_frame.rowconfigure(0, weight=1)

        # Set the font for the Treeview
        style = ttk.Style()
        style.configure("Treeview", font=('Arial', 14))  # Set font size to 14 for Treeview items
        style.configure("Treeview.Heading", font=('Arial', 14, 'bold'))  # Set font size to 14 and bold for headings
        style.configure("Treeview", rowheight=25)

        self.case_list = ttk.Treeview(self.case_list_frame, columns=("Case ID", "Threshold", "Pattern"), show="headings", height=10, style="Treeview")
        self.case_list.heading("Case ID", text="Case ID")
        self.case_list.heading("Threshold", text="Threshold")
        self.case_list.heading("Pattern", text="Pattern")
        self.case_list.column("Case ID", width=160)  # Adjusted width
        self.case_list.column("Threshold", width=100)  # Adjusted width
        self.case_list.column("Pattern", width=500)  # Adjusted width
        self.case_list.bind("<<TreeviewSelect>>", self.on_case_select)
        self.case_list.grid(row=0, column=0, sticky="nsew")

        self.case_list_scroll_x = ctk.CTkScrollbar(self.case_list_frame, orientation="horizontal", command=self.case_list.xview)
        self.case_list_scroll_x.grid(row=1, column=0, sticky="ew")

        self.case_list_scroll_y = ctk.CTkScrollbar(self.case_list_frame, orientation="vertical", command=self.case_list.yview)
        self.case_list_scroll_y.grid(row=0, column=1, sticky="ns")

        self.case_list.configure(xscrollcommand=self.case_list_scroll_x.set, yscrollcommand=self.case_list_scroll_y.set)

        self.event_details_frame = ctk.CTkFrame(self.frame)
        self.event_details_frame.grid(row=6, column=0, columnspan=3, padx=5, pady=5, sticky="nsew")
        self.event_details_frame.grid_remove()

        # Make the event_details_frame expandable
        self.event_details_frame.columnconfigure(0, weight=1)
        self.event_details_frame.rowconfigure(1, weight=1)

        self.event_details_header = ctk.CTkLabel(self.event_details_frame, text="Case Events", font=("Arial", 14, "bold"))
        self.event_details_header.grid(row=0, column=0, sticky="w")

        self.event_details_text = ctk.CTkTextbox(self.event_details_frame, width=760, height=10, wrap="none", font=("Arial", 12))  # Increased width to match the table
        self.event_details_text.grid(row=1, column=0, padx=5, pady=5, sticky="nsew")

        self.event_details_scrollbar_y = ctk.CTkScrollbar(self.event_details_frame, orientation="vertical", command=self.event_details_text.yview)
        self.event_details_scrollbar_y.grid(row=1, column=1, sticky="ns")
        self.event_details_text.configure(yscrollcommand=self.event_details_scrollbar_y.set)

        self.event_details_scrollbar_x = ctk.CTkScrollbar(self.event_details_frame, orientation="horizontal", command=self.event_details_text.xview)
        self.event_details_scrollbar_x.grid(row=2, column=0, sticky="ew")
        self.event_details_text.configure(xscrollcommand=self.event_details_scrollbar_x.set)

        self.show_gantt_chart_button = ctk.CTkButton(self.frame, text="Show Gantt Chart (Selected Case)", command=self.show_gantt_chart, font=("Arial", 12))
        self.show_gantt_chart_button.grid(row=7, column=0, padx=5, pady=5, sticky="w")
        self.show_gantt_chart_button.grid_remove()  # Initially hidden
        self.show_gantt_chart_button_ttp = self.create_tooltip(self.show_gantt_chart_button, "Opens a Gantt Chart visualizing the currently selected case, alongside detected patterns.")

        self.petri_net_frame = ctk.CTkFrame(self.frame)
        self.petri_net_frame.grid(row=8, column=0, columnspan=2, padx=5, pady=5, sticky="nsew")
        self.petri_net_frame.grid_remove()

        # Make the petri_net_frame expandable
        self.petri_net_frame.columnconfigure(0, weight=1)
        self.petri_net_frame.rowconfigure(1, weight=1)

        self.petri_net_header = ctk.CTkLabel(self.petri_net_frame, text="Petri Net", font=("Arial", 12, "bold"))
        self.petri_net_header.grid(row=0, column=0, sticky="ew")

        self.petri_net_canvas = ctk.CTkCanvas(self.petri_net_frame)
        self.petri_net_canvas.grid(row=1, column=0, padx=5, pady=5, sticky="nsew
            ")

        self.h_scrollbar = ctk.CTkScrollbar(self.petri_net_frame, orientation="
            horizontal", command=self.petri_net_canvas.xview)
        self.h_scrollbar.grid(row=2, column=0, sticky="ew")
        self.v_scrollbar = ctk.CTkScrollbar(self.petri_net_frame, orientation="
            vertical", command=self.petri_net_canvas.yview)
        self.v_scrollbar.grid(row=1, column=1, sticky="ns")

        self.petri_net_canvas.configure(xscrollcommand=self.h_scrollbar.set,
            yscrollcommand=self.v_scrollbar.set)

        self.event_log = None
```

**Listing 7.7:** Classification Algorithm: __init__

Listing 7.8 shows the functions that are called to show a tooltip and to show the help-window that can be brought up by clicking the question mark symbol in the GUI.

```
class CollaborationPatternApp:
    def create_tooltip(self, widget, text):
        tool_tip = ctk.CTkToplevel(widget)
        tool_tip.wm_overrideredirect(True)
        tool_tip.wm_geometry("+0+0")
        tool_tip.withdraw()
        label = ctk.CTkLabel(tool_tip, text=text, justify='left', wraplength
            =300)
        label.pack(ipadx=1)
        def enter(event):
            x, y, _, _ = widget.bbox("insert")
            x += widget.winfo_rootx() + 25
            y += widget.winfo_rooty() + 25
            tool_tip.wm_geometry(f"+{x}+{y}")
            tool_tip.deiconify()
        def leave(event):
            tool_tip.withdraw()
        widget.bind("<Enter>", enter)
        widget.bind("<Leave>", leave)
        return tool_tip

    def show_help(self):
        help_window = ctk.CTkToplevel(self.root)
        help_window.title("Help")
        help_text = ("Software used to identify collaborative patterns in event
            log.\n"
                     "Requirements in the event log (exact names):\n\n"
                     "case_id, activity, start_timestamp, end_timestamp,
                         resource\n\n"
                     "Created by Benjamin Meier, University of Regensburg,
                         2024.\n"
                     "E-Mail: Benjamin_paul_meier@yahoo.de")
        help_label = ctk.CTkLabel(help_window, text=help_text, justify='left',
            padx=10, pady=10)
        help_label.pack(padx=10, pady=10)
```

**Listing 7.8:** Classification Algorithm: create_tooltip, show_help

Once the user clicks the GUI-button to load an event log, the "load_event_log" function is called. It opens a file explorer window, which allows the user to select the desired .CSV file that needs to be analyzed. After the user opened a file, the program gets the currently inserted threshold value and opens the event log from the file, using the previously described read_event_log function. Afterwards, the events are grouped by their events (=activities) and sorted by their Case_ID, but only if the "case_id" field is a sortable integer value. Once those preliminary tasks have been performed, the program creates a computing thread for each case within the list of cases, which calls the "process_case" function for each case. Once all threads are finished, the list of results is sorted by case_id again, and stored in a list (sorted_cases). At the end of the load_event_log function, the list of cases, the "View Process Model" button, the "Analyze Process Variants" button and the "Save as CSV" button are shown. The "process_case" function is used to find patterns for one case, and calls the find_patterns_in_case to do so.

```
class CollaborationPatternApp:
    def load_event_log(self):
        # Called if GUI Button is clicked
        global file_path
        file_path = filedialog.askopenfilename(
            title="Open Event Log",
            filetypes=[("CSV files", "*.csv"), ("All files", "*.*")]
        )
        if not file_path:
            return

        try:
            # Get threshold value
            threshold_percentage = self.threshold_entry.get()
            threshold_percentage = float(threshold_percentage) if
                threshold_percentage.isdigit() else 30.0

            # Open event log from file
            event_log = read_event_log(file_path)

            # Group events by cases
            self.case_patterns = defaultdict(dict)
            self.contributing_activities = defaultdict(lambda: defaultdict(list)
                )
            self.case_thresholds = {}

            case_events_dict = defaultdict(list)
            for event in event_log:
                case_events_dict[event['case_id']].append(event)

            # Sort cases by case_id: integer IDs numerically, do not sort non-
                integer Case IDs ("Case-A")
            def case_id_key(case_id):
                try:
                    return (0, int(case_id))
                except ValueError:
                    return (1, str(case_id))

            sorted_case_events_dict = dict(sorted(case_events_dict.items(), key=
                lambda item: case_id_key(item[0])))
```

```python

            threads = []
            # Create one thread for each case (calls "process_case" method for each case)
            for case_id, case_events in sorted_case_events_dict.items():
                thread = threading.Thread(target=self.process_case, args=(case_id, case_events, threshold_percentage))
                threads.append(thread)
                thread.start()

            # Wait for all threads to finish
            for thread in threads:
                thread.join()

            self.event_log = event_log
            self.case_list.delete(*self.case_list.get_children())

            # Gather cases and patterns into a list and sort it
            sorted_cases = sorted(self.case_patterns.items(), key=lambda item: case_id_key(item[0]))
            for case_id, patterns in sorted_cases:
                pattern_names = ', '.join(set(patterns.values()))
                threshold = self.case_thresholds[case_id].total_seconds()  # Use the threshold value from the case
                self.case_list.insert("", "end", values=(case_id, f"{threshold:.2f} s", pattern_names))

            self.event_details_frame.grid_remove()
            self.petri_net_frame.grid_remove()

            # Show the view_model_button, save_button, and case_list_frame when the event log is loaded
            self.view_model_button.grid()
            self.analyze_variants_button.grid()
            self.save_button.grid()
            self.case_list_frame.grid()
        except Exception as e:
            messagebox.showerror("Error", str(e))

    def process_case(self, case_id, case_events, threshold_percentage):
        # Find patterns, save found patterns and contributing activities
        case_id, patterns, contributing_acts, threshold = find_patterns_in_case(case_id, case_events, threshold_percentage)
        self.case_patterns[case_id] = patterns
        self.contributing_activities[case_id] = contributing_acts
        self.case_thresholds[case_id] = threshold
```

**Listing 7.9:** Classification Algorithm: load_event_log, process_case

The functions "on_case_select" and "display_case_events" are called if the user clicks on a particular case in the list of cases. The GUI then shows all activities of the selected case in another part of the window, alongside their patterns and the contributing activities. Additionally, the user is shown a button to create a Gantt-chart for the currently selected case.

```python
class CollaborationPatternApp:
    def on_case_select(self, event):
        # If user clicks on a single case
        selected_item = self.case_list.selection()
        if selected_item:
            case_id = self.case_list.item(selected_item[0], "values")[0]
            self.display_case_events(case_id)
            # Show the show_gantt_chart_button when a case is selected
            self.show_gantt_chart_button.grid()

    def display_case_events(self, case_id):
        case_events = [event for event in self.event_log if event['case_id'] ==
            case_id]
        patterns = self.case_patterns[case_id]
        contributing_activities = self.contributing_activities[case_id]

        self.event_details_text.delete(1.0, "end")
        headers = "Case ID, Activity, Start Timestamp, End Timestamp, Resource,
            Pattern, Contributing Activities"
        self.event_details_text.insert("end", headers + "\n")
        self.event_details_text.insert("end", "-" * len(headers) + "\n")

        for event in case_events:
            pattern = patterns.get(event['activity'], "")
            contributing_acts = ', '.join(contributing_activities[event['
                activity']]) if pattern in ['Collaboration', 'Cooperation']
                else ''
            event_details = f"{event['case_id']}, {event['activity']}, {event['
                start_timestamp']}, {event['end_timestamp']}, {event['resource
                ']}, {pattern} {f'( {contributing_acts} )' if contributing_acts
                 else ''}"
            self.event_details_text.insert("end", event_details + "\n")
        self.event_details_frame.grid()
```

**Listing 7.10:** Classification Algorithm: on_case_select, display_case_events

Listing 7.11 shows the function to display a Petri net of the currently imported event log. It uses the Inductive Miner of the PM4PY library to discover a process tree of the current log and then converts it to a petri net. It also modifies the "decorations" object list of the petri net to assign unique random colors to each operator/resource performing the activities. This is achieved by iterating though the "decoration" objects, and updating the labels and colors based on resource types and activities. The modified "decorations" list is then used by "gviz" to generate a colored image of the petri net, which is then opened.

```
    def generate_random_color(self):
        return "#{:06x}".format(random.randint(0, 0xFFFFFF))

    def display_petri_net(self):
        # Check if the event log is loaded
        if self.event_log is None:
            messagebox.showerror("Error", "No event log loaded.")
            return

        try:
            # Convert event log to a DataFrame
            df = pd.DataFrame(self.event_log)

            # Format the DataFrame for pm4py
```

```python
            log = pm4py.format_dataframe(df, case_id='case_id', activity_key='activity', timestamp_key='start_timestamp')

            # Discover the process tree from the event log
            process_tree = pm4py.discover_process_tree_inductive(log)

            # Convert the process tree to a Petri net
            net, im, fm = pm4py.convert_to_petri_net(process_tree)

            # Get decorations for the Petri net visualization
            decorations = pm4py.visualization.petri_net.util.alignments_decoration.get_alignments_decoration(net, im, fm, log=log, aligned_traces=None, parameters=None)

            # Common color for activities performed by multiple resources
            common_color = 'green'

            # DataFrame to track activities and their associated resources
            dfAlreadyAdded = pd.DataFrame(columns=["activity_name", "resource"])

            # Dictionary to store colors for resources
            resource_colors = {}

            # Assign a random color to each unique resource
            unique_resources = df['resource'].unique()
            for resource in unique_resources:
                resource_colors[resource] = self.generate_random_color()

            # Iterate over the decorations to update labels and colors
            for key, value in decorations.items():
                if 'label' in value:
                    labelVal = value['label']
                    label_without_brackets = labelVal.split('(')[0].strip()

                    # Get unique resources associated with the activity
                    resources = df[df['activity'] == label_without_brackets]['resource'].unique()
                    resources_list = ", ".join(resources)

                    # Iterate over the rows in the DataFrame
                    for index, row in df.iterrows():
                        activity_name = row['activity']
                        resourceval = row['resource']

                        # Assign colors based on resource type
                        if label_without_brackets == activity_name:
                            new_label = f"{resources_list}: {activity_name} ({value['count_move_on_model']},{value['count_fit']})"
                            value['label'] = new_label

                            # Check if the activity-resource pair is already added
                            result = dfAlreadyAdded[dfAlreadyAdded["activity_name"] == activity_name]

                            if not result.empty:
                                if not (result['resource'] == resourceval).any():
                                    value['color'] = common_color
                                    new_label = f"{resources_list}: {activity_name} ({value['count_move_on_model']},{value['count_fit']})"
                                    value['label'] = new_label
                            else:
                                value['color'] = resource_colors[resourceval]
                                new_row = {"activity_name": activity_name, "resource": resourceval}
                                dfAlreadyAdded = pd.concat([dfAlreadyAdded, pd.DataFrame([new_row])], ignore_index=True)

            # Apply decorations and visualize the Petri net
            gviz = pn_visualize.apply(net, im, fm, decorations=decorations)
            image = pn_visualizer.view(gviz)

            # Open the image file and read the image data
            with open(image, 'rb') as img_file:
                img_data = img_file.read()
                self.petri_net_image = ImageTk.PhotoImage(data=img_data)

        except Exception as e:
            # Print and display any error that occurs
            print("Error", str(e))
            messagebox.showerror("Error", str(e))
```

**Listing 7.11:** Classification Algorithm: display_petri_net

The next function, "show_gantt_chart", is called by the user either after inspecting a single case or from the "Analyze Process Variants" window. It first retrieves the currently selected case (if existent) and then filters all relevant events from the event log. The dataframe containing those events is then cleaned by removing activities that do not contain start- and end-timestamps. Afterwards, the dataframe is sorted by their start timestamp in ascending order. The module "Plotly Express" is then used in order to create the Gantt Chart itself. Subsequently, annotations for the discovered patterns are added to the Gantt Chart. This is achieved by retrieving the previously discovered patterns for each activity as well as their contributing activities. If the pattern is either "Collaboration" or "Cooperation", the contributing activities are included in the annotation, but not for the other possible patterns. Each annotation is also colored according to its pattern. At the end of the function, the figure is shown, which opens it in a new tab in the system's standard browser. Due to a rare bug of Plotly express that was found during testing, an alternative way of showing the chart has been implemented, but commented out. If the user experiences problems when loading the Gantt charts, they can comment out *fig.show()*, and remove the Hash-symbol on the *figure0.write_html('first_figure.html', auto_open=True)*-line, directly afterwards.

```
class CollaborationPatternApp:
    def show_gantt_chart(self):
        # Draw Gantt chart using plotly (express)

        # Get the selected item from the case list
        selected_item = self.case_list.selection()
        if selected_item:
            case_id = self.case_list.item(selected_item[0], "values")[0]
            # Filter events for the selected case ID from the event log
            case_events = [event for event in self.event_log if event['case_id']
                 == case_id]
            df = pd.DataFrame(case_events)

            # Ensure there are valid timestamps
            df = df.dropna(subset=['start_timestamp', 'end_timestamp'])

            # Sort the DataFrame by start timestamp
            df = df.sort_values(by='start_timestamp')

            # Log the dataframe for debugging
            print(df)

            # Retrieve the threshold value for the selected case
            threshold = self.case_thresholds[case_id].total_seconds()

            # Create a Gantt chart using Plotly
            fig = px.timeline(df, x_start="start_timestamp", x_end="
                end_timestamp", y="activity", color="resource",
                              title=f"Gantt Chart for Case {case_id}, Threshold
                                  value = {threshold:.2f} seconds")

            # Retrieve patterns and contributing activities for the selected
                case
            patterns = self.case_patterns[case_id]
            contributing_activities = self.contributing_activities[case_id]

            # Adding annotations for Collaboration, Cooperation, Synchronized,
                and Coexistence patterns
            annotations = []
            for event in case_events:
                # Get the pattern for the current activity
                pattern = patterns.get(event['activity'], "")
                # Determine the annotation text and background color
                if pattern in ['Collaboration', 'Cooperation']:
                    contrib_acts = ', '.join(contributing_activities[event['
                        activity']])
                    annotation_text = f"{pattern} ({contrib_acts})"
                    bgcolor = "lightblue" if pattern == 'Collaboration' else "
                        lightgreen"
                elif pattern in ['Synchronized', 'Coexistence']:
                    annotation_text = pattern
                    bgcolor = "lightgoldenrodyellow" if pattern == 'Synchronized
                        ' else "lightcoral"

                # Add annotation if pattern is recognized
                if pattern in ['Collaboration', 'Cooperation', 'Synchronized', '
                    Coexistence']:
                    annotations.append(dict(
                        x=event['end_timestamp'],
                        y=event['activity'],
                        text=annotation_text,
                        showarrow=True,
                        arrowhead=1,
                        ax=-20,
                        ay=-40,
                        bgcolor=bgcolor
                    ))

            fig.update_layout(annotations=annotations)

            # Ensure the first activity is at the top of the chart
            sorted_activities = df['activity'].unique()
            fig.update_yaxes(categoryorder='array', categoryarray=
                sorted_activities)

            fig.update_layout(xaxis_title="Time", yaxis_title="Activity")

            # Attempt to show the figure
            try:
                fig.show()
                # If issues occur when showing the Gannt Chart:
                # Comment out fig.show, and remove the # on the next line:
                #figure0.write_html('first_figure.html', auto_open=True)
            except Exception as e:
                messagebox.showerror("Error", f"Failed to display Gantt chart.
                    Error: {str(e)}")
```

**Listing 7.12:** Classification Algorithm: show_gantt_chart

The function "save_as_csv" saves the currents list of cases and Patterns as a .csv file for further analysis. It first asks the user to select a file path where they want to save the file, and afterwards write the values into this file.

```
class CollaborationPatternApp:
    def save_as_csv(self):
        if not self.event_log:
            return

        file_path = filedialog.asksaveasfilename(
            defaultextension=".csv",
            filetypes=[("CSV files", "*.csv"), ("All files", "*.*")],
        )
        if not file_path:
            return

        try:
            with open(file_path, mode='w', newline='') as file:
                writer = csv.writer(file)
                headers = ["Case ID", "Threshold", "Pattern"]
                writer.writerow(headers)
                for item in self.case_list.get_children():
                    values = self.case_list.item(item, "values")
                    writer.writerow(values)
            messagebox.showinfo("Success", "File saved successfully!")
        except Exception as e:
            messagebox.showerror("Error", str(e))
```

**Listing 7.13:** Classification Algorithm: save_as_csv

Listing 7.14 shows the code for the analysis of process variants. The method "discover_process_variants" iterates through the event log and stores all unique combinations of activities as well as all individual combinations of identified patterns, alongside their quantity, in dictionaries. The "analyze_process_variants" calls this "discover_process_variants" function, and the opens a new window which displays a list of all discovered process variants. For each variant, all discovered combinations of patterns are displayed. The share of the variant in respect to the total number of cases is shown, and the share of each combination of patterns within one variant. The variants are sorted in descending order by their frequency within the event log. If one variant has a number of pattern combinations, the variant itself is shown only once. This is achieved by simulating merged cells, which means hiding duplicate entries in the "Variant" column. Additionally, the first row of each variant is colored in blue, to increase the readability.

```
class CollaborationPatternApp:
    def discover_process_variants(self, event_log):
        # Dictionary to store lists of activities for each case
        self.case_variants = defaultdict(list)

        # Dictionary to key-value-store the count of each pattern for each
            variant and the case IDs
        variant_patterns = defaultdict(lambda: defaultdict(int))
        variant_case_ids = defaultdict(lambda: defaultdict(list))

        # Iterate over each event in the event log
        for event in event_log:
            # Append the activity of the current event to the list of activities
                 for the corresponding case
            self.case_variants[event['case_id']].append(event['activity'])

        # Iterate over each case and its activities
        for case_id, activities in self.case_variants.items():
            # Create a variant string by joining activities with '>'
            variant = '>'.join(activities)
            # Get the patterns for the current case
            patterns = self.case_patterns[case_id].values()
            # Create a string of patterns joined by ', '
            pattern_str = ', '.join(patterns)
            # Increment the count for this pattern in the variant_patterns
                dictionary
            variant_patterns[variant][pattern_str] += 1
            # Store the case ID in the variant_case_ids dictionary
            variant_case_ids[variant][pattern_str].append(case_id)

        # Return the dictionaries containing variants, their pattern counts, and
            case IDs
        return variant_patterns, variant_case_ids

    def analyze_process_variants(self):
        # Ensure the event log is loaded
        if not self.event_log:
            return

        # Discover the process variants from the event log
        process_variants, variant_case_ids = self.discover_process_variants(self
            .event_log)

        # Create a new window to display the process variants
```

```python
        variants_window = ctk.CTkToplevel(self.root)
        variants_window.title("Process Variants")
        variants_window.geometry("1500x800")  # Set the initial size of the window

        # Create a treeview widget to display the process variants
        tree = ttk.Treeview(variants_window, columns=("Index", "Variant", "Share (total)", "Pattern", "Percentage"), show="headings", height=20)
        tree.heading("Index", text="Index", anchor='w')
        tree.heading("Variant", text="Variant", anchor='w')
        tree.heading("Share (total)", text="Share (total)", anchor='w')
        tree.heading("Pattern", text="Pattern", anchor='w')
        tree.heading("Percentage", text="Percentage", anchor='w')
        tree.grid(row=0, column=0, sticky="nsew")

        # Add scrollbars to the treeview
        scroll_y = ctk.CTkScrollbar(variants_window, orientation="vertical", command=tree.yview)
        scroll_y.grid(row=0, column=1, sticky="ns")
        tree.configure(yscrollcommand=scroll_y.set)

        scroll_x = ctk.CTkScrollbar(variants_window, orientation="horizontal", command=tree.xview)
        scroll_x.grid(row=1, column=0, sticky="ew")
        tree.configure(xscrollcommand=scroll_x.set)

        # Configure the grid to make the treeview expandable
        variants_window.grid_rowconfigure(0, weight=1)
        variants_window.grid_columnconfigure(0, weight=1)

        # Calculate the total number of instances across all variants
        total_instances = sum(sum(details.values()) for details in process_variants.values())

        tree_data = []
        for index, (variant, details) in enumerate(process_variants.items(), start=1):
            variant_total_instances = sum(details.values())
            variant_share = (variant_total_instances / total_instances) * 100
            variant_rows = []
            for pattern, count in details.items():
                percentage = (count / variant_total_instances) * 100
                variant_rows.append((index, variant, f"{variant_share:.2f}%", pattern, percentage))
            # Sort variant rows by percentage (descending order)
            variant_rows.sort(key=lambda x: -x[4])
            tree_data.extend(variant_rows)

        # Sort tree data by share (descending order) and by percentage within each variant
        tree_data.sort(key=lambda x: (-float(x[2].strip('%')), x[0], -x[4]))

        # Renumber the indices to always start from 1
        current_index = 0
        previous_variant = None

        # Insert sorted data into the tree
        for row_data in tree_data:
            variant = row_data[1]
            if variant != previous_variant:
                current_index += 1
                previous_variant = variant
            item = tree.insert("", "end", values=(current_index, variant, row_data[2], row_data[3], f"{row_data[4]:.2f}%"))
            # Highlight the most common pattern line
            if row_data[4] == max(row[4] for row in tree_data if row[1] == row_data[1]):
                tree.item(item, tags=("highlight",))

        # Define the tag configuration for highlighting
        tree.tag_configure("highlight", background="lightblue")

        # Set column widths
        max_variant_width = int(self.root.winfo_screenwidth() * 0.4)
        for col in tree["columns"]:
            max_width = font.Font().measure(col)
            for item in tree.get_children():
                cell_value = tree.item(item)["values"][tree["columns"].index(col)]
                col_width = font.Font().measure(cell_value)
                if max_width < col_width:
                    max_width = col_width
            if col == "Variant":
                tree.column(col, width=min(max_variant_width, max_width))
            else:
                tree.column(col, width=max_width)

        # Simulate merged cells by hiding text in duplicate entries
        self.variant_rows = {}
        prev_variant = None
        for item in tree.get_children():
            current_values = tree.item(item)["values"]
            if prev_variant and prev_variant == current_values[1]:  # Check if the variant is the same as the previous one
                tree.set(item, column="Variant", value="")
                tree.set(item, column="Index", value="")
                tree.set(item, column="Share (total)", value="")
            else:
                prev_variant = current_values[1]
            self.variant_rows[item] = current_values  # Store the actual values for each row

        tree.bind("<Double-1>", lambda event: self.on_pattern_double_click(event, tree, variant_case_ids))
```

**Listing 7.14:** Classification Algorithm: analyze_process_variants

The following three functions are used to analyze which cases match the exact process variant and pattern combination in the list. If the user double clicks on a particular combination of patterns, thew function "show_cases_for_variant" is called, which displays a list of all such cases in a new window. If the user then double clicks on one of those cases, a Gantt chart will be opened for this case, similar to pressing the "Show Gantt chart (Current case)" button in the main window.

```
class CollaborationPatternApp:
    def on_pattern_double_click(self, event, tree, variant_case_ids):
        selected_item = tree.selection()
        if selected_item:
            # Retrieve the actual values from the stored dictionary
            values = self.variant_rows[selected_item[0]]
            variant = values[1]
            pattern = values[3]
            self.show_cases_for_variant(variant, pattern, variant_case_ids)

    def show_cases_for_variant(self, variant, pattern, variant_case_ids):
        cases_window = ctk.CTkToplevel(self.root)
        cases_window.title(f"Cases for Variant: {variant} with Pattern: {pattern
            }")
        cases_window.geometry("400x400")

        # Create a treeview widget to display the cases
        tree = ttk.Treeview(cases_window, columns=("CaseID",), show="headings",
            height=20)
        tree.heading("CaseID", text="Case ID", anchor='w')
        tree.grid(row=0, column=0, sticky="nsew")

        # Add scrollbars to the treeview
        scroll_y = ctk.CTkScrollbar(cases_window, orientation="vertical",
            command=tree.yview)
        scroll_y.grid(row=0, column=1, sticky="ns")
        tree.configure(yscrollcommand=scroll_y.set)

        scroll_x = ctk.CTkScrollbar(cases_window, orientation="horizontal",
            command=tree.xview)
        scroll_x.grid(row=1, column=0, sticky="ew")
        tree.configure(xscrollcommand=scroll_x.set)

        # Configure the grid to make the treeview expandable
        cases_window.grid_rowconfigure(0, weight=1)
        cases_window.grid_columnconfigure(0, weight=1)

        # Insert matching cases into the tree
        for case_id in variant_case_ids[variant][pattern]:
            tree.insert("", "end", values=(case_id,))

        # Bind click event to show Gantt chart for selected case
        tree.bind("<Double-1>", lambda event: self.on_case_id_double_click(event
            , tree))

    def on_case_id_double_click(self, event, tree):
        selected_item = tree.selection()
        if selected_item:
            case_id = tree.item(selected_item[0], "values")[0]
            self.show_gantt_chart_for_case(case_id)
```

**Listing 7.15:** Classification Algorithm: on_pattern_double_click, show_cases_for_variant, on_case_id_double_click

Listing 7.16 shows the main function of the program, which is loaded when the user opens the program. It sets certain parameters about the GUI such as the appearance mode and color theme, and then creates a continuously running loop for the "CollaborationPatternApp" class, which encapsulates the GUI of the program and much of its other functionality.

```
if __name__ == "__main__":
    ctk.set_appearance_mode("Light")  # Modes: "System" (default), "Dark", "
        Light"
    ctk.set_default_color_theme("dark-blue")  # Themes: "blue" (default), "green
        ", "dark-blue"
    root = ctk.CTk()
    app = CollaborationPatternApp(root)
    root.mainloop()
```

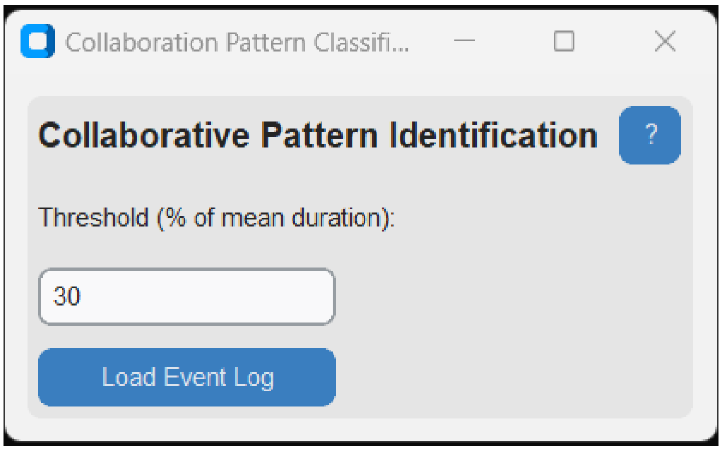
**Listing 7.16:** Classification Algorithm: main Function

## 7.5 Classification Algorithm Usage Guide

The following section explains how to use the program in detail so that users can operate it optimally.

### 7.5.1 Loading Event Logs

After following the steps of 7.2.3 to open the program, the user will see the following window.


Collaboration Pattern Classifi...
Collaborative Pattern Identification

Threshold (% of mean duration):
30

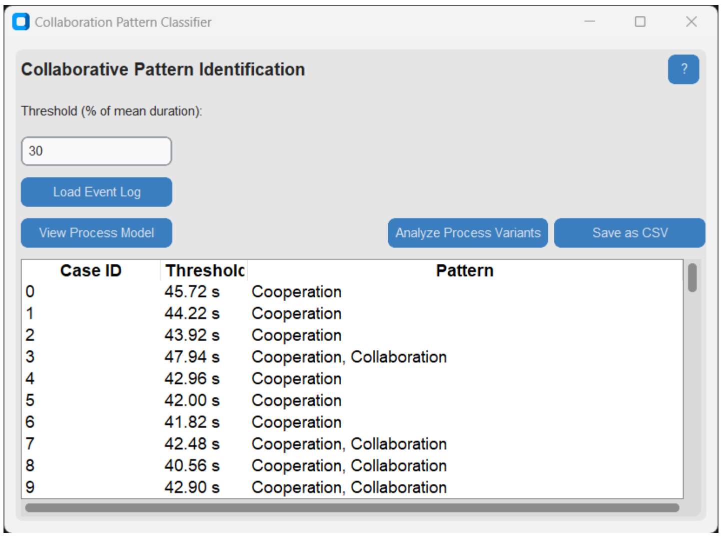
**Figure 7.1:** Starting screen of the program

The user then has the option to change the Threshold value (as described in 6.2.1) to a different value than the predefined one, which is 30% of the mean duration of activities. After setting the threshold value accordingly or leaving it at its default value, the user can click "Load Event Log". This opens a file explorer window which allows the user to select the event log to be analyzed. It must be ensured that the event log fulfills the requirements in 7.3. If this is not the case, individual attributes of the event log must be renamed accordingly, using spreadsheet software or text editors.

### 7.5.2 Analyzing individual Cases

After opening the event log, the program might take some time to process the event log, depending on the length of the log and the performance of the hardware used.

The implementation uses multi-threaded computing in order to speed up the detection process. Once the analysis is complete, the GUI will show a list of all cases, alongside the case-specific threshold value and a list of all patterns that were detected within a case.

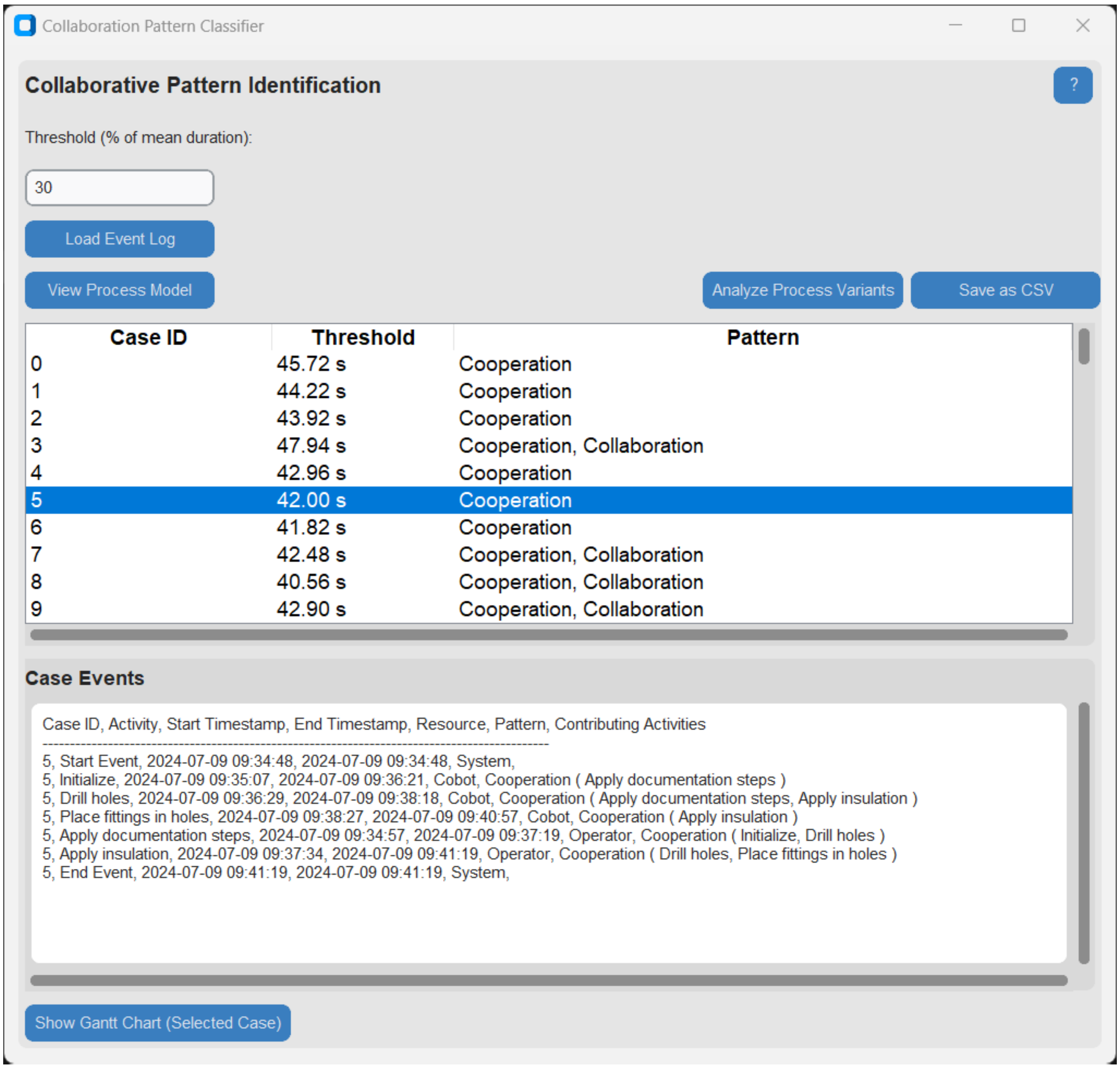

Collaboration Pattern Classifier
Collaborative Pattern Identification
?
Load Event Log
View Process Model
Analyze Process Variants
Save as CSV

Threshold (% of mean duration): 30

| Case ID | Thresholc | Pattern |
|---|---|---|
| 0 | 45.72 s | Cooperation |
| 1 | 44.22 s | Cooperation |
| 2 | 43.92 s | Cooperation |
| 3 | 47.94 s | Cooperation, Collaboration |
| 4 | 42.96 s | Cooperation |
| 5 | 42.00 s | Cooperation |
| 6 | 41.82 s | Cooperation |
| 7 | 42.48 s | Cooperation, Collaboration |
| 8 | 40.56 s | Cooperation, Collaboration |
| 9 | 42.90 s | Cooperation, Collaboration |

**Figure 7.2:** Program view after opening an event log

The user can then select any of the cases in the list by clicking on it. Once this is done, the window shows information about each individual activity of the case, such as all attributes of one activity, as well as the determined collaborative pattern 7.3. If the pattern is either "Cooperation" or "Collaboration", it will also show which other activities contributed towards this classification. In the example below, the activity "Drill holes" has the pattern "Cooperation", and the activities "Apply documentation steps" and "Apply insulation" counted towards it.


Collaboration Pattern Classifier
Collaborative Pattern Identification
?
Load Event Log
View Process Model
Analyze Process Variants
Save as CSV
Case Events
Show Gantt Chart (Selected Case)

Threshold (% of mean duration): 30

| Case ID | Threshold | Pattern |
|---|---|---|
| 0 | 45.72 s | Cooperation |
| 1 | 44.22 s | Cooperation |
| 2 | 43.92 s | Cooperation |
| 3 | 47.94 s | Cooperation, Collaboration |
| 4 | 42.96 s | Cooperation |
| 5 | 42.00 s | Cooperation |
| 6 | 41.82 s | Cooperation |
| 7 | 42.48 s | Cooperation, Collaboration |
| 8 | 40.56 s | Cooperation, Collaboration |
| 9 | 42.90 s | Cooperation, Collaboration |

```
Case ID, Activity, Start Timestamp, End Timestamp, Resource, Pattern, Contributing Activities
-------------------------------------------------------------------------------------------
5, Start Event, 2024-07-09 09:34:48, 2024-07-09 09:34:48, System,
5, Initialize, 2024-07-09 09:35:07, 2024-07-09 09:36:21, Cobot, Cooperation ( Apply documentation steps )
5, Drill holes, 2024-07-09 09:36:29, 2024-07-09 09:38:18, Cobot, Cooperation ( Apply documentation steps, Apply insulation )
5, Place fittings in holes, 2024-07-09 09:38:27, 2024-07-09 09:40:57, Cobot, Cooperation ( Apply insulation )
5, Apply documentation steps, 2024-07-09 09:34:57, 2024-07-09 09:37:19, Operator, Cooperation ( Initialize, Drill holes )
5, Apply insulation, 2024-07-09 09:37:34, 2024-07-09 09:41:19, Operator, Cooperation ( Drill holes, Place fittings in holes )
5, End Event, 2024-07-09 09:41:19, 2024-07-09 09:41:19, System,
```

**Figure 7.3:** Program showing information about a single case

### 7.5.3 Viewing Gantt Charts about individual Cases

After selecting an individual case, it is possible to show a Gantt chart depicting the the sequence of activities within a case, as well as the patterns found. After the "Show Gantt Chart (Selected Case)" button is clicked, a window will open with the standard internet browser of the system, and a chart will be displayed, as seen in figure 7.4. The user can hover over a single activity within the Gantt Chart to view all attributes of the activity.

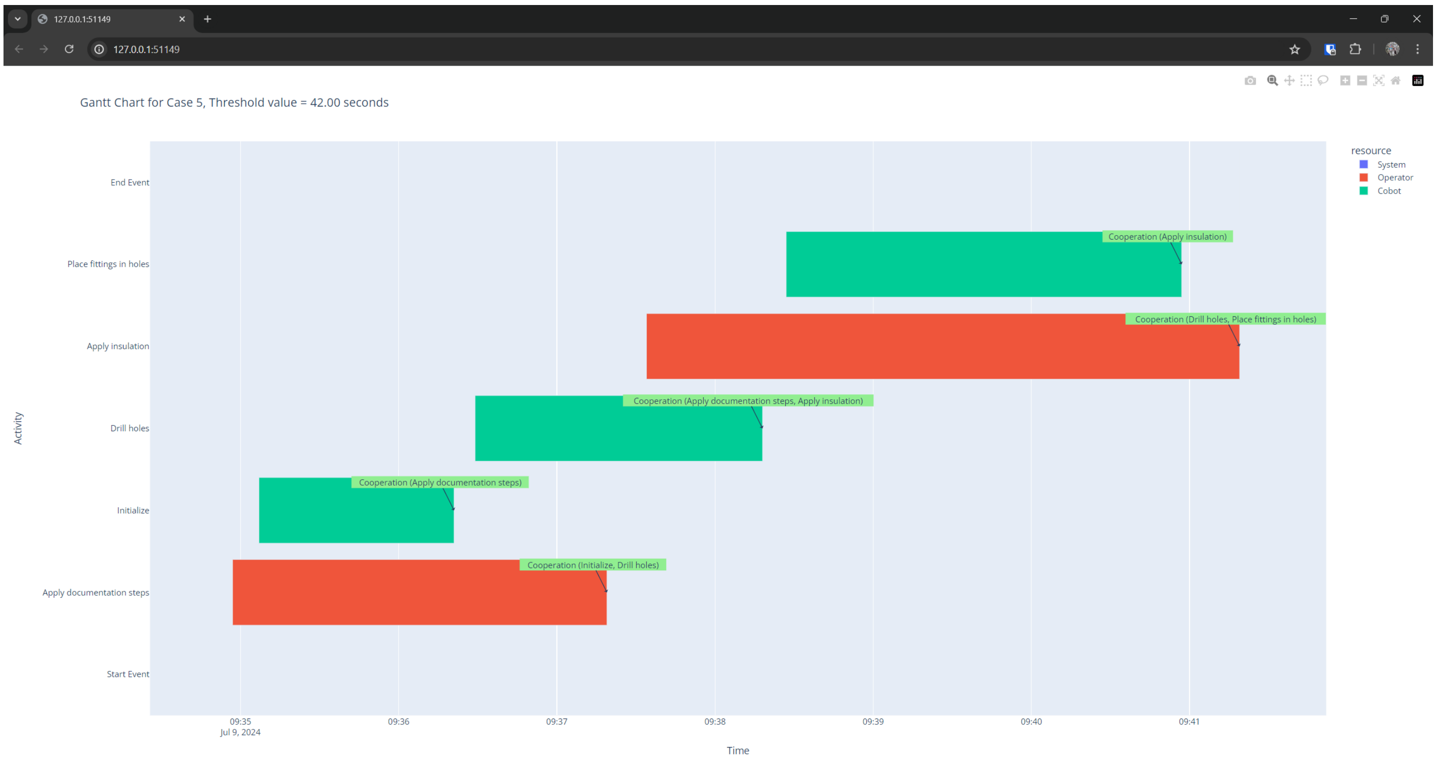
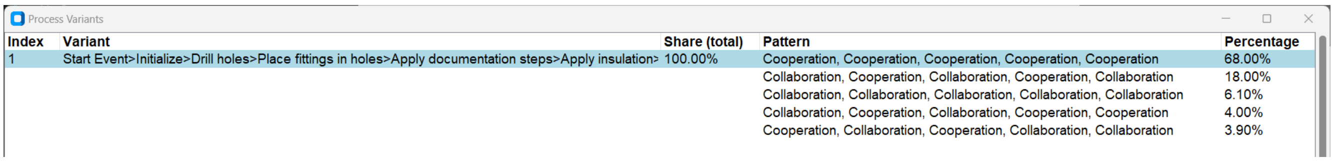


**Figure 7.4:** A Gantt chart about a single case

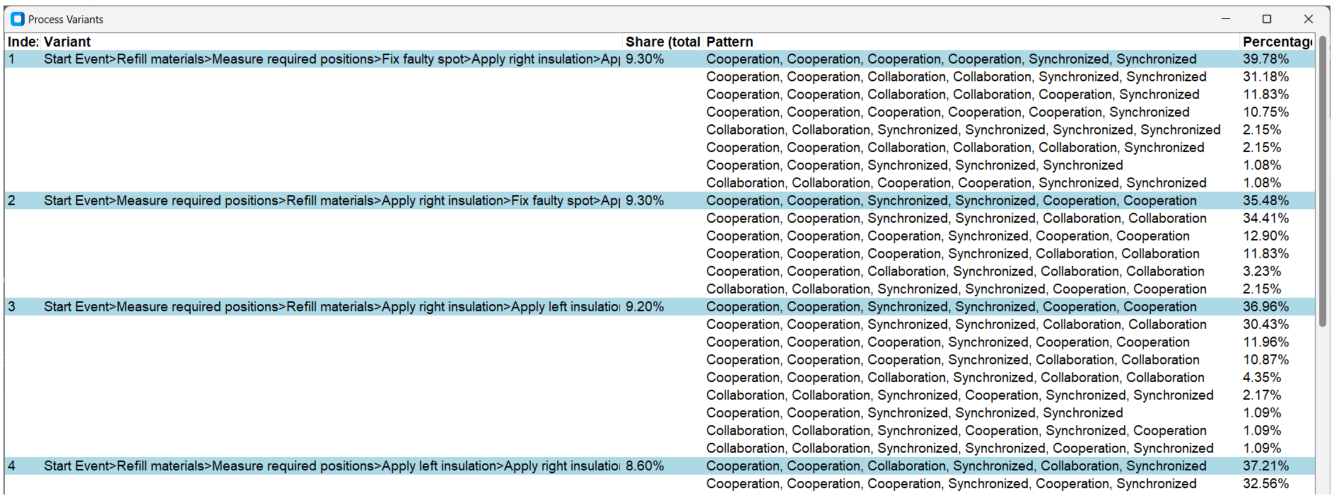
## 7.5.4 Analyzing Process Variants

Within the main window of the program, the user can click the button "Analyze Process variants" in order to open a new window. This window lists all existing process variants, as well as the frequency of each pattern classification for each process variant. A percentage is shown next to the discovered patterns. The user can then make an informed decision which combination of detected patterns is the most likely for each process variant, supported by the displayed percentage values.



| Index | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Start Event>Initialize>Drill holes>Place fittings in holes>Apply documentation steps>Apply insulation> | 100.00% | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation | 68.00% |
| | | | Collaboration, Cooperation, Collaboration, Cooperation, Collaboration | 18.00% |
| | | | Collaboration, Collaboration, Collaboration, Collaboration, Collaboration | 6.10% |
| | | | Collaboration, Cooperation, Collaboration, Cooperation, Cooperation | 4.00% |
| | | | Cooperation, Collaboration, Cooperation, Collaboration, Collaboration | 3.90% |

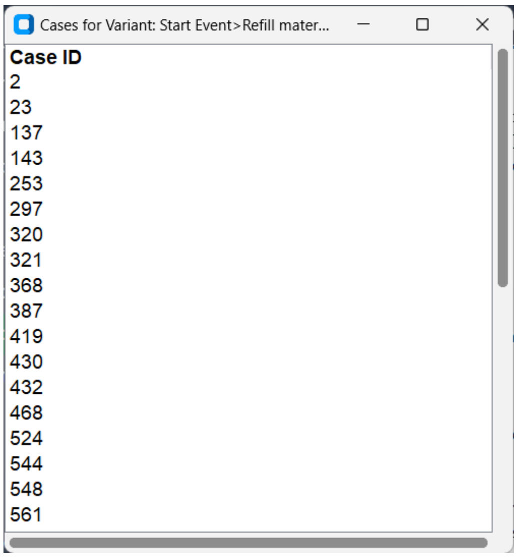
**Figure 7.5:** Process variants window if there is only one variant


Process Variants

| Inde: | Variant | Share (total | Pattern | Percentag |
|---|---|---|---|---|
| 1 | Start Event>Refill materials>Measure required positions>Fix faulty spot>Apply right insulation>Ap | 9.30% | Cooperation, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 39.78% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Synchronized, Synchronized | 31.18% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Cooperation, Synchronized | 11.83% |
| | | | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation, Synchronized | 10.75% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized | 2.15% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Collaboration, Synchronized | 2.15% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 1.08% |
| | | | Collaboration, Collaboration, Cooperation, Cooperation, Synchronized, Synchronized | 1.08% |
| 2 | Start Event>Measure required positions>Refill materials>Apply right insulation>Fix faulty spot>Ap | 9.30% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 35.48% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 34.41% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 12.90% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 11.83% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 3.23% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Cooperation | 2.15% |
| 3 | Start Event>Measure required positions>Refill materials>Apply right insulation>Apply left insulatio | 9.20% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 36.96% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 30.43% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 11.96% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 10.87% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 4.35% |
| | | | Collaboration, Collaboration, Synchronized, Cooperation, Synchronized, Synchronized | 2.17% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 1.09% |
| | | | Collaboration, Collaboration, Synchronized, Cooperation, Synchronized, Cooperation | 1.09% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Synchronized | 1.09% |
| 4 | Start Event>Refill materials>Measure required positions>Apply left insulation>Apply right insulatio | 8.60% | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Synchronized | 37.21% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Synchronized | 32.56% |

**Figure 7.6:** Process variants window if there is more than one variant

If the user double-clicks on a combination of patterns within the process variants window, a list of all cases of the selected Process Variant/Patterns combination will be shown:


Cases for Variant: Start Event>Refill mater...

| Case ID |
|---|
| 2 |
| 23 |
| 137 |
| 143 |
| 253 |
| 297 |
| 320 |
| 321 |
| 368 |
| 387 |
| 419 |
| 430 |
| 432 |
| 468 |
| 524 |
| 544 |
| 548 |
| 561 |

**Figure 7.7:** List of cases belonging to a certain process variant/ patterns combination

The user can then double-click any of the cases within the list to display the Gantt chart of the selected case, similar to 7.5.3.

### 7.5.5 Additional Functionality

In this subsection, additional functionality of the implementation that are not core functionality are described.

#### Displaying Process Models

The user can create and view process models by clicking "View Process Model" The program then creates a Process model using the inductive mining algorithm [63]. To further improve the transparency of the work transfer between individual resources, a separate (randomly selected) color is selected for each actor. Figure 7.8 shows an exemplary process model discovered by the program. It must be noted that the Inductive Mining algorithm does not take the end-timestamps of the event log into account, which means that parallel activities may be displayed in a misleading way. In order to view the actual sequence of activities, a per-case analysis is necessary, supported by the possibility of viewing Gantt charts, as described in 7.5.3.

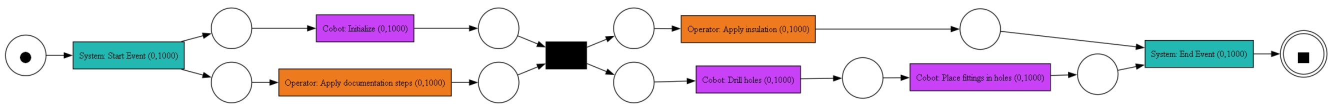

System: Start Event (0,1000)
Cobot: Initialize (0,1000)
Operator: Apply documentation steps (0,1000)
Operator: Apply insulation (0,1000)
Cobot: Drill holes (0,1000)
Cobot: Place fittings in holes (0,1000)
System: End Event (0,1000)

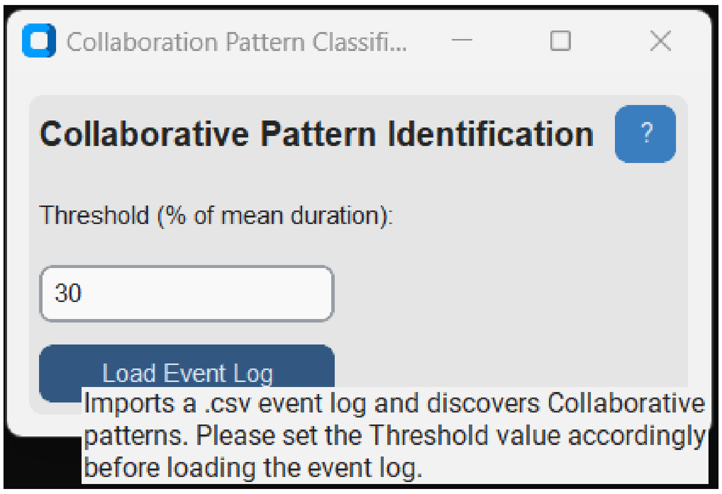
**Figure 7.8:** A process model discovered by the inductive mining algorithm

### Exporting Cases lists

By clicking the "Save as CSV" button, the user can export the current list of cases and their detected patterns as a CSV file. This allows for further analysis using other means, such as spreadsheet software.

### Displaying help text

By clicking the Question-mark symbol, the user will be shown additional information about the program. By hovering over certain elements of the GUI, the user will be shown tooltips, which show helpful information regarding the current element.


Collaboration Pattern Classifi...
Collaborative Pattern Identification
?
Threshold (% of mean duration):
30
Load Event Log

Imports a .csv event log and discovers Collaborative patterns. Please set the Threshold value accordingly before loading the event log.

**Figure 7.9:** Tooltips within the program

# Chapter 8

# Implementation of the Event Log Generator

Due to the focus of this work being the identification of Collaborative Patterns in event logs, the following sections will only give a short overview over the technical implementation of the event log generator that was written for this purpose.

## 8.1 Implementation Description

The event log generator was implemented using Python, using the *Customtkinter* library for the GUI and standard libraries such as *datetime*, *random*, *csv*, and *os* for core functionalities. The following are key components of the implementation:

- **Activity Simulation:** The implementation simulates the execution of activities by generating random durations based on expected values and variance.
- **Process Flow Handling:** The function to generate a process flow manages the execution sequence of activities, handling XOR and parallel gateways, and logging each activity's start and end times.
- **Event Log Generation:** Another function creates the event logs by simulating multiple process flows across different cases, ensuring variability and realism.
- **File Management:** The event log generator allows for the saving of the generated logs to CSV files, allowing easy modifications and compatibility with the algorithm of this work as well as existing Process Mining tools.
- **GUI:** A user-friendly interface was developed using *Customtkinter*, allowing the user to input parameters, load process definitions, and generate event logs.

## 8.2 Implementation in Python of the Event Log Generator

In this section, similar to the description of the implementation of the Classification algorithm in 7.4, the Python implementation of the log generator software will be laid

out. Again, each important function of the program will be described. Again, please note that the order in which program functions are explained in this section is not necessarily the order in which they are called during the runtime of the program.
The first listing shows the import of all required libraries.

```python
import customtkinter as ctk
from tkinter import filedialog, messagebox
import datetime
import random
import csv
import os
```

**Listing 8.1:** Log Generator: Import of Libraries

Listing 8.2 contains the methods "generate_runtime" and "log_activity". Function "generate_runtime" is only used to generate the actual duration (=runtime) of a generated activity. The function "log_activity" logs details of an activity for a given case into the event log. It starts by checking whether "activity_name" is a tuple (necessary for complex activities such as parallel activities) or just text, in which case it looks for the resource in "resource_mapping". Next, it searches for the activity in the "activities" list to determine the required duration. It then uses this value and variance to randomly generate the actual runtime. The activity is then added to the event log.

```python
def generate_runtime(expected_value, variance):
    return random.gauss(expected_value, variance)

def log_activity(case_id, activity_name, start_time, activities,
    resource_mapping, event_log, delay):
    if isinstance(activity_name, tuple):
        activity_name, resource = activity_name
    else:
        resource = resource_mapping.get(activity_name, 'Unknown')

    activity = next((a for a in activities if a['name'] == activity_name), None)
    if activity is None:
        print(f"Warning: Activity '{activity_name}' not found in the activities
            list.")
        return start_time

    runtime = generate_runtime(activity['expected_value'], activity['variance'])
    start_timestamp = start_time.strftime('%Y-%m-%d %H:%M:%S')
    end_timestamp = (start_time + datetime.timedelta(seconds=runtime)).strftime
        ('%Y-%m-%d %H:%M:%S')
    event_log.append({
        'case_id': case_id,
        'activity': activity['name'],
        'start_timestamp': start_timestamp,
        'end_timestamp': end_timestamp,
        'resource': resource
    })
    return start_time + datetime.timedelta(seconds=runtime)
```

**Listing 8.2:** Log Generator: generate_runtime, log_activity

Listing 8.3 describes the function to handle XOR gateways in the process flow. The function starts by checking if the index of the activity ("i") is present in the "xor_probabilities" dictionary. If the index is found, the function selects one of the possible activities from the dictionaries based on the probabilities. The chosen activity is then returned.

```python
def handle_xor_gateway(activity_name, xor_probabilities, i):
    if i in xor_probabilities:
        return random.choices(activity_name[1:], weights=xor_probabilities[i])
            [0]
    else:
        print(f"Warning: XOR gateway index '{i}' not found in the
            xor_probabilities dictionary.")
        return activity_name[1]
```

**Listing 8.3:** Log Generator: handle_xor_gateway

The next function, "generate_process_flow" (listing 8.4), is used for generating a single case. After initializing the "start_time", the function iterates over each "activity_name" in "process_flows". If "activity_name" is a tuple and contains "XOR", it calls the "handle_xor_gateway" function, seen in 8.3. If "activity_name" is a tuple and contains "PARALLEL", it handles multiple activities that should run in parallel. It does so by setting the "parallel_start_time" to the current "start_time" and tracking the parallel end times. If the "random_order" flag is set to "True", the order of the parallel activities is shuffled. It then iterates through each "parallel_activity_name", determines the appropriate start time (based on the resource availability), and logs the activity. "resource_end_times" are updated to ensure that the same resource does not start a new activity before the previous one ends. In any case (even if no XOR or PARALLEL gateway needs to be handled), the activity is logged and the "start_time" is updated. The last part of the function handles looping. It does so by checking whether the activity has a "loop_probability" and logs the activity multiple times based on the loop probability.

```python
def generate_process_flow(case_id, process_flow, activities, resource_mapping,
    xor_probabilities, event_log, random_order, delay):
    start_time = datetime.datetime.now()
    i = 0
    for activity_name in process_flow:
        if isinstance(activity_name, tuple) and 'XOR' in activity_name:
            activity_name = handle_xor_gateway(activity_name, xor_probabilities,
                 i)
            i += 1

        if isinstance(activity_name, tuple) and 'PARALLEL' in activity_name:
            parallel_start_time = start_time
            parallel_end_times = []
            resource_end_times = {}
            if random_order:
                parallel_activities = list(activity_name[1:])
                random.shuffle(parallel_activities)
            else:
                parallel_activities = list(activity_name[1:])

            for parallel_activity_name in parallel_activities:
                resource = resource_mapping.get(parallel_activity_name, 'Unknown')
                if resource in resource_end_times:
                    # Ensure the next activity for the same resource starts after the previous one ends
                    activity_start_time = resource_end_times[resource]
                else:
                    # Add a random delay to the first activity's start time
                    activity_start_time = parallel_start_time + datetime.timedelta(seconds=random.uniform(0, delay))

                end_time = log_activity(case_id, parallel_activity_name, activity_start_time, activities, resource_mapping, event_log, delay)
                parallel_end_times.append(end_time)
                resource_end_times[resource] = end_time + datetime.timedelta(seconds=random.uniform(0, delay))

            # Update start_time to the latest end_time of parallel activities
            start_time = max(parallel_end_times, default=start_time)
            continue

        # Log activity
        start_time = log_activity(case_id, activity_name, start_time, activities, resource_mapping, event_log, delay)

        # Handle looping
        activity = next((a for a in activities if a['name'] == activity_name), None)
        if activity and 'loop_probability' in activity:
            while random.random() < activity['loop_probability']:
                start_time = log_activity(case_id, activity_name, start_time, activities, resource_mapping, event_log, delay)
```

**Listing 8.4:** Log Generator: generate_process_flow

The next listing 8.5 shows the functions "generate_event_log" and "save_event_log". The short function "generate_event_log" has the purpose of generating individual cases for each process flow. The function "save_event_log" is used for file handling, meaning that it generates the output CSV file and saves the event log.

```
def generate_event_log(num_cases, process_flows, activities, resource_mapping, xor_probabilities, random_order, delay):
    event_log = []
    for case_id in range(num_cases):
        process_flow = random.choice(process_flows)
        generate_process_flow(case_id, process_flow, activities, resource_mapping, xor_probabilities, event_log, random_order, delay)
    return event_log

def save_event_log(event_log):
    now = datetime.datetime.now()
    formatted_date = now.strftime('%d%m')
    formatted_time = now.strftime('%H%M')

    # Define the output directory and ensure it exists
    output_dir = 'Output'
    os.makedirs(output_dir, exist_ok=True)

    # Define the filename with the output directory
    filename = os.path.join(output_dir, f"e_log_{formatted_date}_{formatted_time
        }.csv")

    with open(filename, 'w', newline='') as csvfile, open("Output\\log.txt", 'w
        ', newline='') as logFile:
        fieldnames = ['case_id', 'activity', 'start_timestamp', 'end_timestamp',
             'resource']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        writer.writeheader()
        for event in event_log:
            writer.writerow(event)

    return filename
```

**Listing 8.5:** Log Generator: generate_event_log, save_event_log

Listing 8.6 is a function called by the GUI once the user clicks the "Generate Event Log" button. It gets all the parameters (number of cases, random order, delay) and then calls the other functions "generate_event_log" and "save_event_log". The other function in this listing, "browse_file", is used to open a file browser so that the user can select an input file.

```
def generate_log():
    num_cases = int(cases_entry.get())
    random_order = random_order_var.get()
    delay = float(delay_entry.get())

    algorithm_code = algorithm_text.get("1.0", "end")

    # Use a dictionary to execute the code
    local_vars = {}
    exec(algorithm_code, globals(), local_vars)

    activities = local_vars.get('activities', [])
    resource_mapping = local_vars.get('resource_mapping', {})
    xor_probabilities = local_vars.get('xor_probabilities', [])
    process_flows = local_vars.get('process_flows', [])

    event_log = generate_event_log(num_cases, process_flows, activities,
        resource_mapping, xor_probabilities, random_order, delay)
    filename = save_event_log(event_log)

    # Show confirmation message
    messagebox.showinfo("Confirmation", f"Event log successfully generated and
        saved to '{filename}'")

def browse_file():
    filename = filedialog.askopenfilename(filetypes=[("Text files", "*.txt")])
    if filename:
        with open(filename, 'r') as file:
            algorithm_text.delete("1.0", "end")
            algorithm_text.insert("1.0", file.read())
```

**Listing 8.6:** Log Generator: generate_log, browse_file

The last listing, 8.7, shows all code that is required in order to show the GUI for the Log generator.

```python
ctk.set_appearance_mode("Light")  # Modes: "System" (default), "Dark", "Light"
ctk.set_default_color_theme("dark-blue")  # Themes: "blue" (default), "green", "
    dark-blue"
root = ctk.CTk()
root.title("Event Log Generator")

algorithm_frame = ctk.CTkFrame(root)
algorithm_frame.pack(fill=ctk.BOTH, expand=True, padx=10, pady=10)

algorithm_label = ctk.CTkLabel(algorithm_frame, text="Input:")
algorithm_label.pack(pady=5)

algorithm_text = ctk.CTkTextbox(algorithm_frame, wrap="word", height=200, width
    =800)
algorithm_text.pack(pady=5)

browse_button = ctk.CTkButton(algorithm_frame, text="Browse", command=
    browse_file)
browse_button.pack(pady=5)

cases_label = ctk.CTkLabel(algorithm_frame, text="Number of Cases:")
cases_label.pack(pady=5)

cases_entry = ctk.CTkEntry(algorithm_frame)
cases_entry.pack(pady=5)

delay_label = ctk.CTkLabel(algorithm_frame, text="Max Delay (seconds):")
delay_label.pack(pady=5)

delay_entry = ctk.CTkEntry(algorithm_frame)
delay_entry.pack(pady=5)

random_order_var = ctk.IntVar()
random_order_checkbox = ctk.CTkCheckBox(algorithm_frame, text="Random Order",
    variable=random_order_var)
random_order_checkbox.pack(pady=5)

generate_button = ctk.CTkButton(algorithm_frame, text="Generate Event Log",
    command=generate_log)
generate_button.pack(pady=20)

root.mainloop()
```

**Listing 8.7:** Log Generator: Parameters of the GUI

## 8.3 Log Generator Usage Guide

This section will describe the steps to use the event log generator to create artificial event logs. Prerequisites for running the algorithm are similar to the Pattern Identification algorithm and are described in section 7.2. The only differences are that the name of the program is "LogGenerator.py", which needs to be run, and only the "customtkinter" library needs to be installed. Screenshot 8.1 shows the screen of the GUI after the program has been opened.

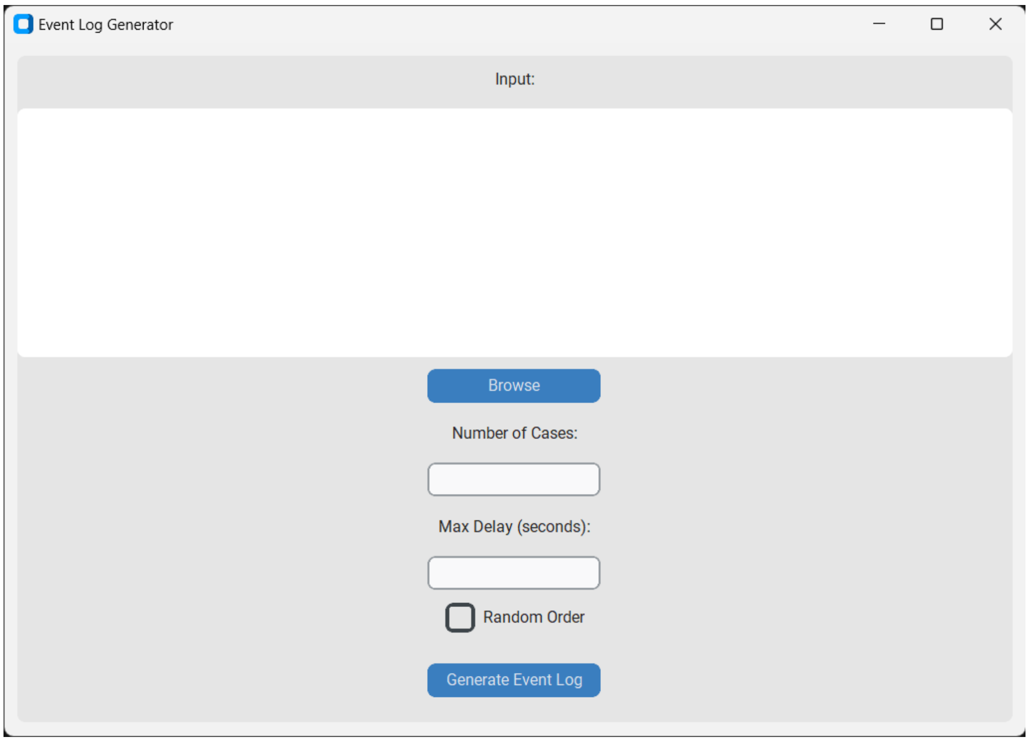


**Figure 8.1:** GUI of the event log generator

At the beginning, the user needs to enter an input file. This can be either achieved by pasting such a file into the window, or using the "Browse" button to import any .txt file containing input data. The contents of input files will be described in more detail later in this chapter.
After inserting an input file, the user needs to set the following options:

- **Number of Cases**: The number of cases which will be present in the created event log.
- **Max delay (seconds)**: Introduces a random delay between activities. If the value is set to zero, an activity will begin right after the previous activity ended. If a positive value is set, the delay will be uniformly distributed between zero and the set value.
- **Random order**: If this check is set, parallel activities will have a randomized order in each case. If they are not set, they will be in the order set in the "Process Flows" in the input. A possible purpose for this are two sequences of activities that run in parallel during the same case, and consist of multiple activities each that need to be in their correct order (Example: Aircraft manufacturing, subsection 5.1.6).

The user then clicks "Generate Event Log". The program now creates the event log and saves it in the "Output" folder, which will be created in the root directory of

the python script. The filename will contain the current day and time, in the format "e_log_mmdd_hhmm".
The input file needs to follow the following format:

- **Activities**: In this listing, all activities need to be included, as well as their expected duration and variance.
- **Resource_Mapping**: In here, the resource performing each activity needs to be included.
- **XOR_probabilities**: If XOR-gateways are present, the probabilities for each one need to be declared. If multiple XOR gateways are present, the declaration with index 0 describes the first-occurring gateway in the process_flows, index 1 the second one, and so on.
- **Process_flows**: In this section, all possible process flows (which equal a possible sequence of activities within a case) need to be declared, as well as parallel and XOR gateways. If a activity needs to be performed by another resource than than the one written in the resource_mapping, it can be written like this:
  ('Activity Name', 'Non-standard-actor')

The following listing shows an example input file containing loops, a parallel gateway and a XOR gateway.

```
activities = [
    {'name': 'Start Event', 'expected_value': 0, 'variance': 0},
    {'name': 'Activity 0', 'expected_value': 80, 'variance': 10, '
        loop_probability': 0.2},
    {'name': 'Activity 1', 'expected_value': 60, 'variance': 10},
    {'name': 'Activity 2', 'expected_value': 120, 'variance': 20, '
        loop_probability': 0.1},
    {'name': 'Activity 3', 'expected_value': 125, 'variance': 20},
    {'name': 'Activity 5', 'expected_value': 90, 'variance': 15},
    {'name': 'Activity 4', 'expected_value': 100, 'variance': 15},
    {'name': 'End Event', 'expected_value': 0, 'variance': 0},
]
resource_mapping = {
    'Start Event': 'System',
    'Activity 0': 'Operator',
    'Activity 1': 'Operator',
    'Activity 2': 'Cobot',
    'Activity 3': 'Operator',
    'Activity 4': 'Cobot',
    'Activity 5': 'Operator',
    'End Event': 'System'
}
xor_probabilities = {0: [0.6, 0.4]}
process_flows = [
    ['Start Event', 'Activity 0', ('XOR', 'Activity 1', 'Activity 2'), 'Activity
         3', ('PARALLEL', 'Activity 5', 'Activity 4'), 'End Event']]
```

**Listing 8.8:** Sample input file for the log generator

The next listing was used to generate the event log for the "Refrigerator assembly" use case. There is only one process flow present, as this process only needs one. The program can, however, also accept multiple process flows to represent different processes within an event log.

```
activities = [
        {'name': 'Start Event', 'expected_value': 0, 'variance': 0},
        {'name': 'Measure required positions', 'expected_value': 150, 'variance': 10},
        {'name': 'Apply left insulation', 'expected_value': 120, 'variance': 10},
        {'name': 'Apply right insulation', 'expected_value': 120, 'variance': 20},
        {'name': 'Refill materials', 'expected_value': 60, 'variance': 10},
        {'name': 'Fix faulty spot', 'expected_value': 100, 'variance': 50},
        {'name': 'Check for leakage', 'expected_value': 120, 'variance': 20},
        {'name': 'End Event', 'expected_value': 0, 'variance': 0},
    ]

resource_mapping = {
        'Start Event': 'System',
        'Measure required positions': 'Cobot',
        'Apply left insulation': 'Cobot',
        'Apply right insulation': 'Cobot',
        'Refill materials': 'Operator',
        'Fix faulty spot': 'Operator',
        'Check for leakage': 'Operator',
        'End Event': 'System'
    }

process_flows = [
    ['Start Event', ('PARALLEL', 'Measure required positions', 'Refill materials'), ('PARALLEL', 'Apply left insulation', 'Apply right insulation', 'Fix faulty spot'), 'Check for leakage', 'End Event']
    ]
```

**Listing 8.9:** "Refrigerator assembly" input file for the log generator

# Chapter 9

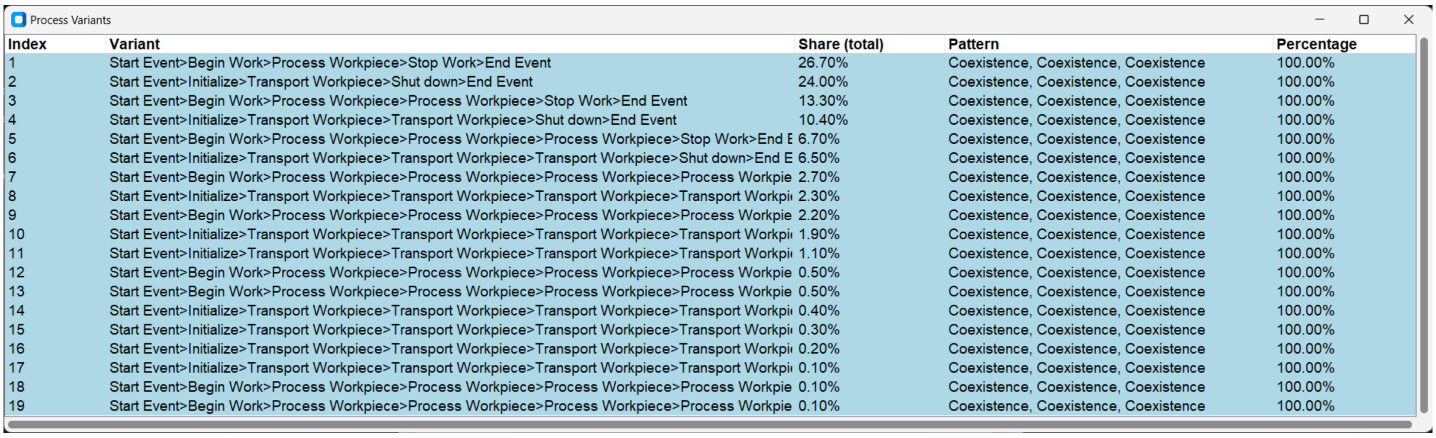
# Evaluation of the Algorithm

## 9.1 Evaluation of Correctness based on Use Cases

In the following section, the implemented algorithm will be used to determine Collaborative Patterns on all use cases previously defined in chapter 5. The threshold value was left at the default value of 30%, as will be described in section 9.2.

### 9.1.1 Evaluation for Pick-and-Place

The use case "Pick an Place" describes a simple "Coexistence" process of an operator and cobot working side-by-side, but not interfering with each other during their workflows. Due to the loops within the process on both the operator and cobot, a large number of process variants (19) was discovered, as can be seen in screenshot 9.1. However, due to the fact that only one resource (either operator or cobot) ever works within a case, the "Coexistence" pattern was always assigned, which is the expected behaviour.

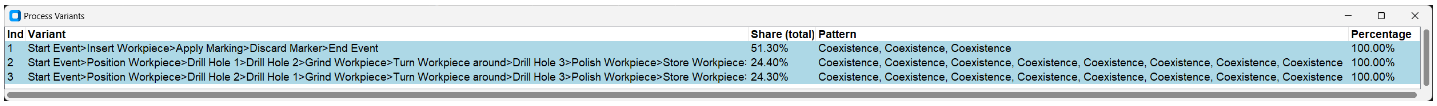

Process Variants

| Index | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Start Event>Begin Work>Process Workpiece>Stop Work>End Event | 26.70% | Coexistence, Coexistence, Coexistence | 100.00% |
| 2 | Start Event>Initialize>Transport Workpiece>Shut down>End Event | 24.00% | Coexistence, Coexistence, Coexistence | 100.00% |
| 3 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Stop Work>End Event | 13.30% | Coexistence, Coexistence, Coexistence | 100.00% |
| 4 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Shut down>End Event | 10.40% | Coexistence, Coexistence, Coexistence | 100.00% |
| 5 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Stop Work>End E | 6.70% | Coexistence, Coexistence, Coexistence | 100.00% |
| 6 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Shut down>End E | 6.50% | Coexistence, Coexistence, Coexistence | 100.00% |
| 7 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 2.70% | Coexistence, Coexistence, Coexistence | 100.00% |
| 8 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 2.30% | Coexistence, Coexistence, Coexistence | 100.00% |
| 9 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 2.20% | Coexistence, Coexistence, Coexistence | 100.00% |
| 10 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 1.90% | Coexistence, Coexistence, Coexistence | 100.00% |
| 11 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 1.10% | Coexistence, Coexistence, Coexistence | 100.00% |
| 12 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 0.50% | Coexistence, Coexistence, Coexistence | 100.00% |
| 13 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 0.50% | Coexistence, Coexistence, Coexistence | 100.00% |
| 14 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 0.40% | Coexistence, Coexistence, Coexistence | 100.00% |
| 15 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 0.30% | Coexistence, Coexistence, Coexistence | 100.00% |
| 16 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 0.20% | Coexistence, Coexistence, Coexistence | 100.00% |
| 17 | Start Event>Initialize>Transport Workpiece>Transport Workpiece>Transport Workpiece>Transport Workpi | 0.10% | Coexistence, Coexistence, Coexistence | 100.00% |
| 18 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 0.10% | Coexistence, Coexistence, Coexistence | 100.00% |
| 19 | Start Event>Begin Work>Process Workpiece>Process Workpiece>Process Workpiece>Process Workpie | 0.10% | Coexistence, Coexistence, Coexistence | 100.00% |

**Figure 9.1:** Process variants for "Pick-and-Place"

### 9.1.2 Evaluation for Hole Drilling

Similar to "Pick and Place", the use case "Hole Drilling" describes a process without any interlocking between the resources active in the process. Only a small number of process variants exist, which all were classified as "Coexistence", which means 100% of the cases were detected correctly.



Process Variants

| Ind | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Start Event>Insert Workpiece>Apply Marking>Discard Marker>End Event | 51.30% | Coexistence, Coexistence, Coexistence | 100.00% |
| 2 | Start Event>Position Workpiece>Drill Hole 1>Drill Hole 2>Grind Workpiece>Turn Workpiece around>Drill Hole 3>Polish Workpiece>Store Workpiece: | 24.40% | Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence | 100.00% |
| 3 | Start Event>Position Workpiece>Drill Hole 2>Drill Hole 1>Grind Workpiece>Turn Workpiece around>Drill Hole 3>Polish Workpiece>Store Workpiece: | 24.30% | Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence, Coexistence | 100.00% |

**Figure 9.2:** Process variants for "Hole Drilling"

### 9.1.3 Evaluation for Glue Application

"Glue application" contains the transfer of work from one actor to another and back again within a case. All cases of the event log are of the same process variant, which means that while durations of activities may vary, the order of operations is always the same within the whole event log. Screenshot 9.3 shows the Gantt chart of an exemplary case. In this chart, it can be seen that there is no temporal overlap between the activities of the Operator (red) and the activity "Apply Glue" of the Cobot (green). For this reason, the correct pattern "Synchronized" is detected in all cases of the event log.

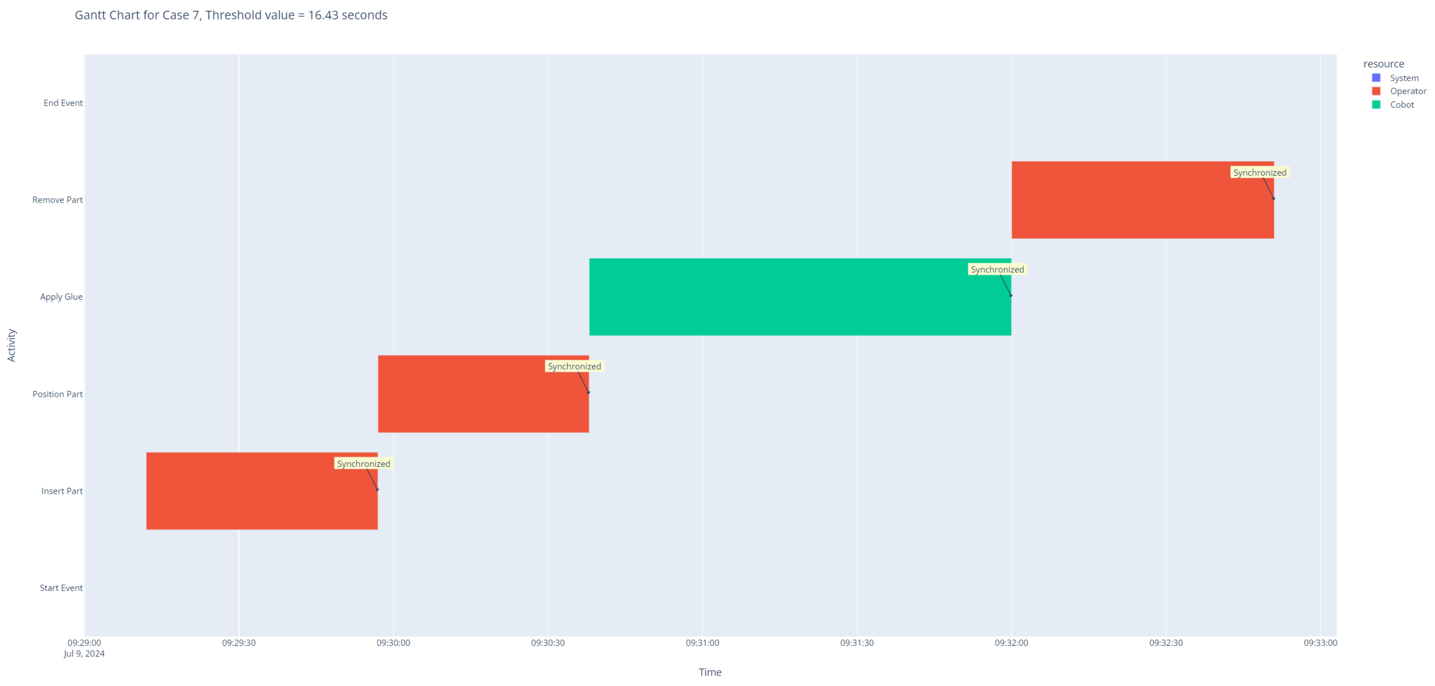
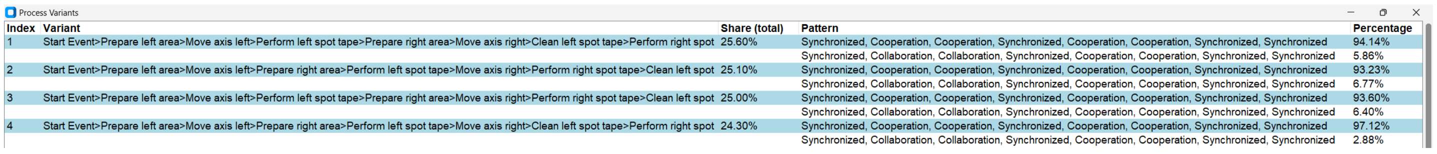


**Figure 9.3:** Exemplary Gantt chart for "Glue Application"

### 9.1.4 Evaluation for Spot Taping

In the "Spot Taping" use case, the operator moves the workpiece to the right or left to perform work on one side of the workpiece, while one of two Cobots performs work on the other side of the workpiece. The algorithm was able to classify 94.53% of all cases correctly, which means a combination of "Cooperation" and "Synchronized", but not "Collaboration". The process variants of the complete "Spot Taping" log are shown in figure 9.4. It can be observed that only a small number of variants is present (4), which means that the workflow within the process has little variation.


Process Variants

| Index | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Start Event>Prepare left area>Move axis left>Perform left spot tape>Prepare right area>Move axis right>Clean left spot tape>Perform right spot | 25.60% | Synchronized, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 94.14% |
| | | | Synchronized, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 5.86% |
| 2 | Start Event>Prepare left area>Move axis left>Prepare right area>Perform left spot tape>Move axis right>Perform right spot tape>Clean left spot | 25.10% | Synchronized, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 93.23% |
| | | | Synchronized, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 6.77% |
| 3 | Start Event>Prepare left area>Move axis left>Perform left spot tape>Prepare right area>Move axis right>Perform right spot tape>Clean left spot | 25.00% | Synchronized, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 93.60% |
| | | | Synchronized, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 6.40% |
| 4 | Start Event>Prepare left area>Move axis left>Prepare right area>Perform left spot tape>Move axis right>Clean left spot tape>Perform right spot | 24.30% | Synchronized, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 97.12% |
| | | | Synchronized, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 2.88% |

**Figure 9.4:** Process variants for "Spot Taping"

Table 9.1 shows the percentage and count of correct classifications within the process variants. As evident from the results, the algorithm achieved a very good performance, classifying 94,53% of all cases correctly.

| Variant | Share (Percentage) | Count | Correct (Percentage) | Correct Cases |
|---|---|---|---|---|
| 1 | 25.6 | 256 | 94.14 | 241 |
| 2 | 25.1 | 251 | 93.23 | 234 |
| 3 | 25.0 | 250 | 93.60 | 234 |
| 4 | 24.3 | 243 | 97.12 | 236 |
| | **100** | **1000** | **94.5** | **945** |

**Table 9.1:** Performance data for "Spot Taping"

Image 9.5 shows a correctly classified instance of "Spot Taping".

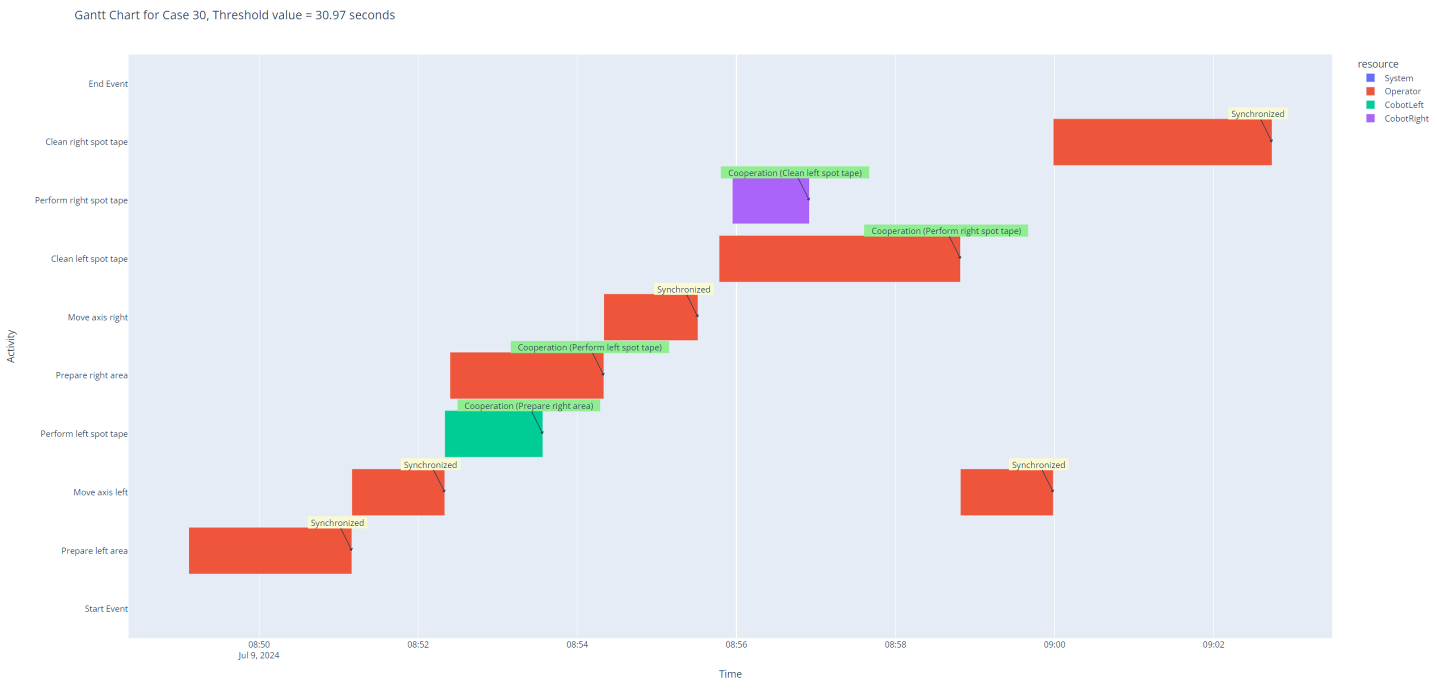


**Figure 9.5:** Correctly classified process variant for "Spot Taping"

Case 264, shown in figure 9.6 shows a wrongly classified process variant. In this case, "Perform left spot tape" falls withing the threshold around the start and slightly within the threshold around the end of "Prepare right area", which means that the algorithm classified it as "Collaboration", more precisely the "Nearly complete Overlap" case, as described in section 6.2.5.

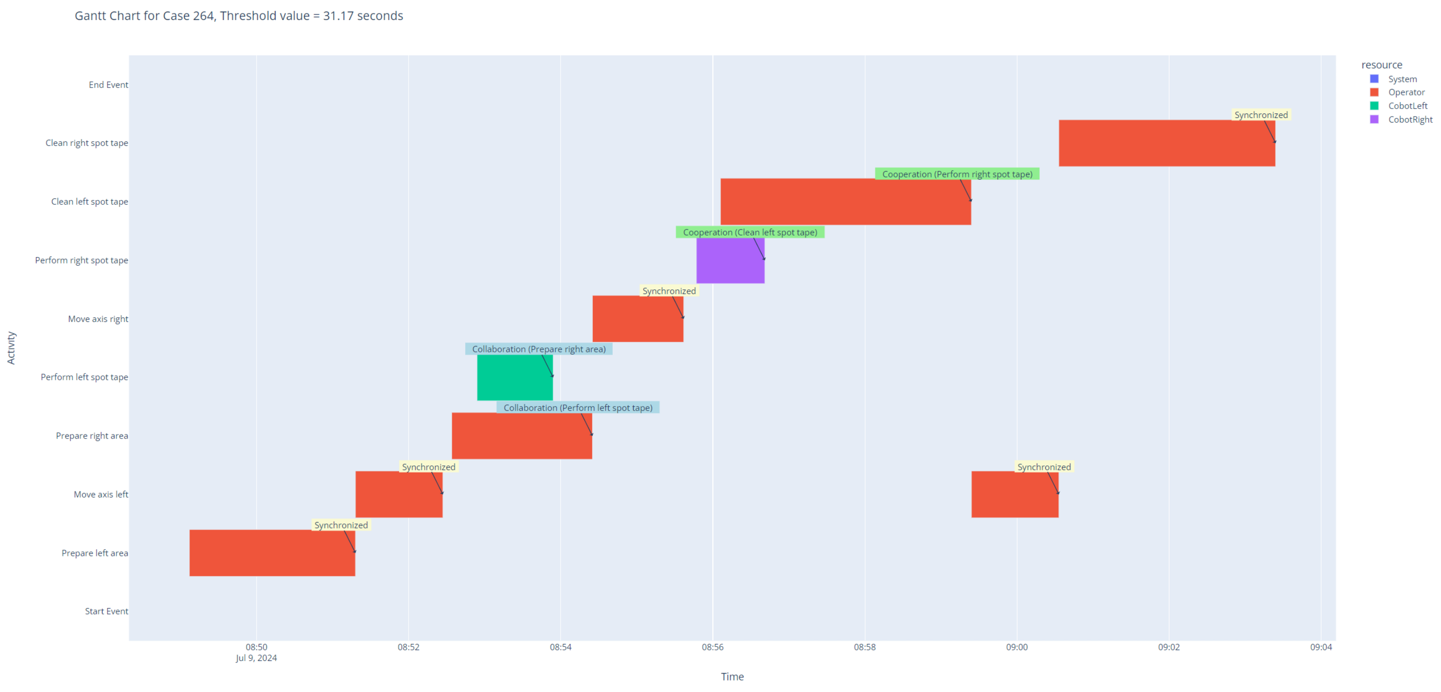
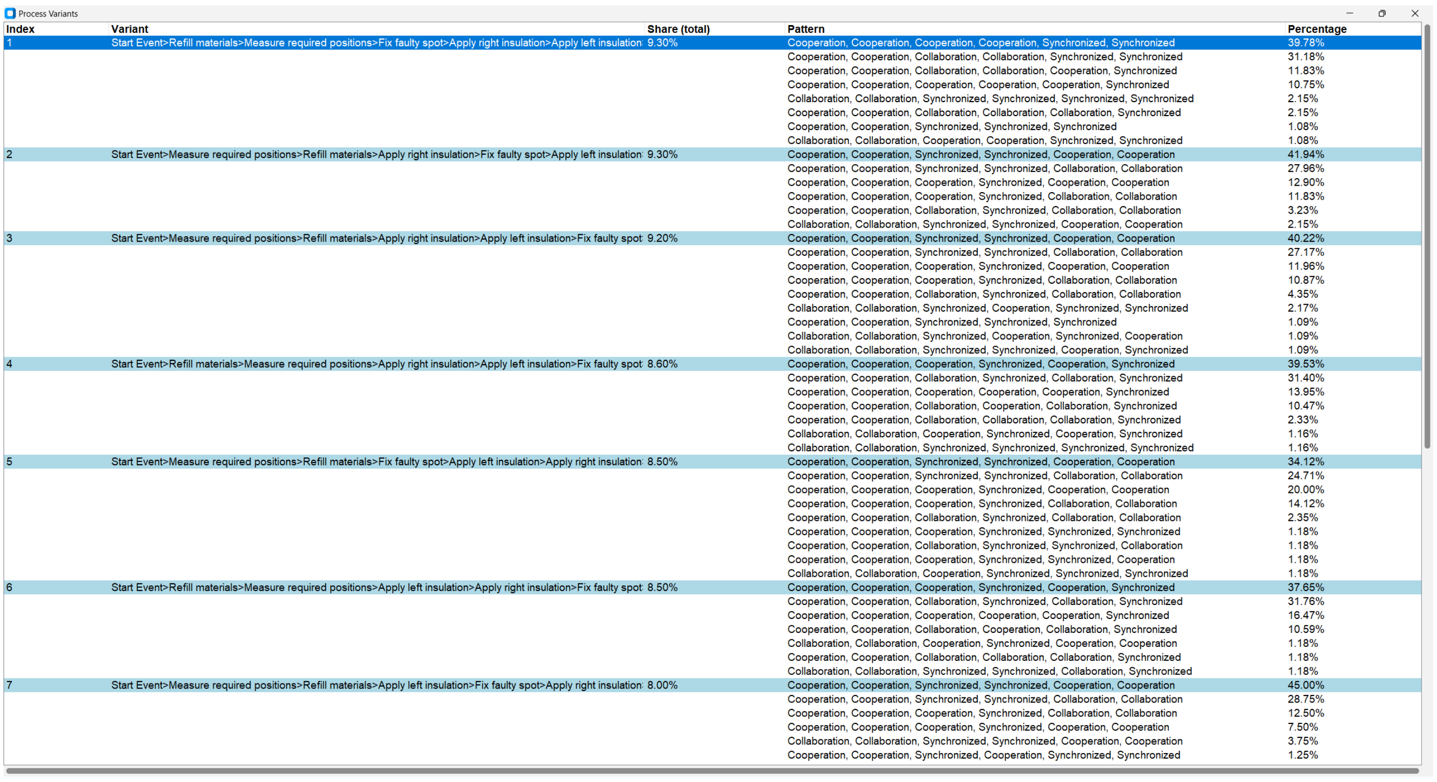


**Figure 9.6:** Wrongly classified process variant for "Spot Taping"

### 9.1.5 Evaluation for Refrigerator Assembly

"Refrigerator Assembly" describes a more complex process of manufacturing refrigerators. It involves parallel work between the actors on the same workpiece, with the cobot performing a number of automated activities while the operator checks the workpiece for faults and fixes them accordingly. Due to the nature of parallel work, a large number of process variants (12) can be observed, ranging from 7.5% to 9.3% share within the total number of cases. Figure 9.7 shows the "Process Variants" screen of the program for this event log. Please note that this list (only within the screenshot) is cut off after variant 7.



| Index | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Start Event>Refill materials>Measure required positions>Fix faulty spot>Apply right insulation>Apply left insulation | 9.30% | Cooperation, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 39.78% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Synchronized, Synchronized | 31.18% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Cooperation, Synchronized | 11.83% |
| | | | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation, Synchronized | 10.75% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized | 2.15% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Collaboration, Synchronized | 2.15% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 1.08% |
| | | | Collaboration, Collaboration, Cooperation, Cooperation, Synchronized, Synchronized | 1.08% |
| 2 | Start Event>Measure required positions>Refill materials>Apply right insulation>Fix faulty spot>Apply left insulation | 9.30% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 41.94% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 27.96% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 12.90% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 11.83% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 3.23% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Cooperation | 2.15% |
| 3 | Start Event>Measure required positions>Refill materials>Apply right insulation>Apply left insulation>Fix faulty spot | 9.20% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 40.22% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 27.17% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 11.96% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 10.87% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 4.35% |
| | | | Collaboration, Collaboration, Synchronized, Cooperation, Synchronized, Synchronized | 2.17% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 1.09% |
| | | | Collaboration, Collaboration, Synchronized, Cooperation, Synchronized, Cooperation | 1.09% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Synchronized | 1.09% |
| 4 | Start Event>Refill materials>Measure required positions>Apply right insulation>Apply left insulation>Fix faulty spot | 8.60% | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Synchronized | 39.53% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Synchronized | 31.40% |
| | | | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation, Synchronized | 13.95% |
| | | | Cooperation, Cooperation, Collaboration, Cooperation, Collaboration, Synchronized | 10.47% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Collaboration, Synchronized | 2.33% |
| | | | Collaboration, Collaboration, Cooperation, Synchronized, Cooperation, Synchronized | 1.16% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized | 1.16% |
| 5 | Start Event>Measure required positions>Refill materials>Fix faulty spot>Apply left insulation>Apply right insulation | 8.50% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 34.12% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 24.71% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 20.00% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 14.12% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 2.35% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 1.18% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Synchronized, Collaboration | 1.18% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Synchronized, Cooperation | 1.18% |
| | | | Collaboration, Collaboration, Cooperation, Synchronized, Synchronized, Synchronized | 1.18% |
| 6 | Start Event>Refill materials>Measure required positions>Apply left insulation>Apply right insulation>Fix faulty spot | 8.50% | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Synchronized | 37.65% |
| | | | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Synchronized | 31.76% |
| | | | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation, Synchronized | 16.47% |
| | | | Cooperation, Cooperation, Collaboration, Cooperation, Collaboration, Synchronized | 10.59% |
| | | | Collaboration, Collaboration, Cooperation, Synchronized, Cooperation, Cooperation | 1.18% |
| | | | Cooperation, Cooperation, Collaboration, Collaboration, Collaboration, Synchronized | 1.18% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Collaboration, Synchronized | 1.18% |
| 7 | Start Event>Measure required positions>Refill materials>Apply left insulation>Fix faulty spot>Apply right insulation | 8.00% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 45.00% |
| | | | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 28.75% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration | 12.50% |
| | | | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 7.50% |
| | | | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Cooperation | 3.75% |
| | | | Cooperation, Cooperation, Synchronized, Cooperation, Synchronized, Synchronized | 1.25% |

**Figure 9.7:** Process variants for "Refrigerator assembly"

The correct classification for this use case is once again a combination of "Synchronized" and "Cooperation", and no "Collaboration". For each variant, the number of cases where this is the case was counted and represented in table 9.2. Depending on the process variant, the algorithm achieved a percentage of correct cases ranging from 51.61 % to 63.64%. Even though the percentage of correct cases was only slightly above 50% in some cases, it was always possible to unambiguously classify the correct pattern combinations for each process variant, since the wrong pattern combinations always had a much smaller percentage in the total number of cases.

| Variant | Share (Percentage) | Count | Correct (Percentage) | Correct Cases |
|---|---|---|---|---|
| 1 | 9.3 | 93 | 51.61 | 48 |
| 2 | 9.3 | 93 | 54.84 | 51 |
| 3 | 9.2 | 92 | 53.27 | 49 |
| 4 | 8.6 | 86 | 53.48 | 46 |
| 5 | 8.6 | 86 | 56.48 | 49 |
| 6 | 8.5 | 85 | 54.12 | 46 |
| 7 | 8.0 | 80 | 53.75 | 43 |
| 8 | 7.9 | 79 | 53.75 | 42 |
| 9 | 7.8 | 78 | 60.25 | 47 |
| 10 | 7.7 | 77 | 63.64 | 49 |
| 11 | 7.6 | 76 | 57.90 | 45 |
| 12 | 7.5 | 75 | 53.34 | 40 |
| | **100** | **1000** | **55.5** | **555** |

**Table 9.2:** Performance data for "Refrigerator Assembly"

An example for this can be seen in table 9.3, which shows all combinations of patterns discovered for process variant 2 in the Refrigerator Assembly use case. The most common sequence of classifications (Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation) was the most common by a large margin (41.94% compared to 27.96% of the next most common variant). Combined, the correct combinations 1 and 3 account for 54.84% of all cases of this variant.

| Combination | Process | Percentage |
|---|---|---|
| 1 | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation | 41.94% |
| 2 | Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration | 27.96% |
| 3 | Cooperation, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 12.90% |
| 4 | Cooperation, Cooperation, Cooperation, Synchronized, Collaboration | 11.83% |
| 5 | Cooperation, Cooperation, Collaboration, Synchronized, Collaboration, Collaboration | 3.23% |
| 6 | Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Cooperation | 2.15% |

**Table 9.3:** Discovered pattern combinations for process variant 2 of "Refrigerator Assembly"

Image 9.8 shows a common process variant of the "Refrigerator assembly" use case, which was classified correctly.

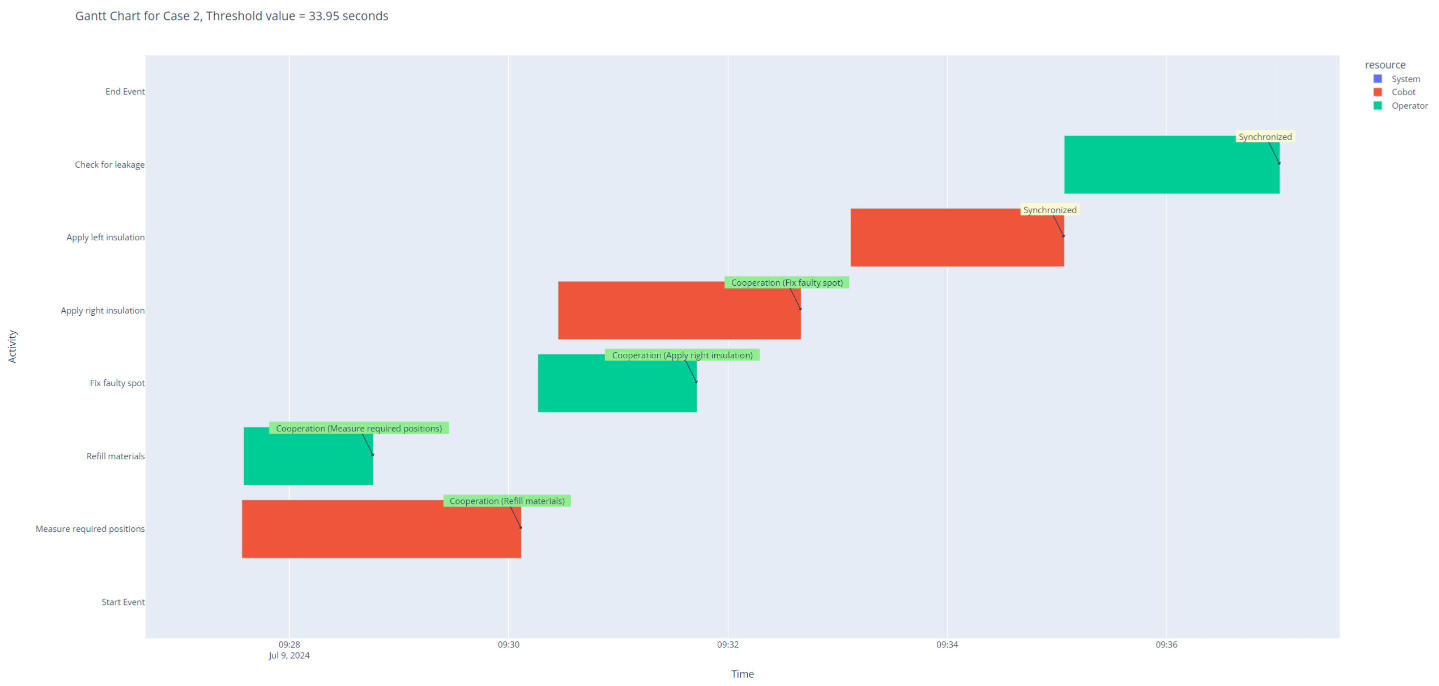


**Figure 9.8:** Correctly classified process variant for "Refrigerator assembly"

Figure 9.9 shows another case , in which the activities "Apply right insulation" and "Fix faulty spot" were incorrectly classified as "Collaboration". This was the case because of the similar start-timestamps and end-timestamps of both activities, leading the algorithm to believe that they are the "Nearly complete Overlap" case (see section 6.2.5). Fortunately this was not a common occurrence, which means that the algorithm still achieved an acceptable result for this use case.

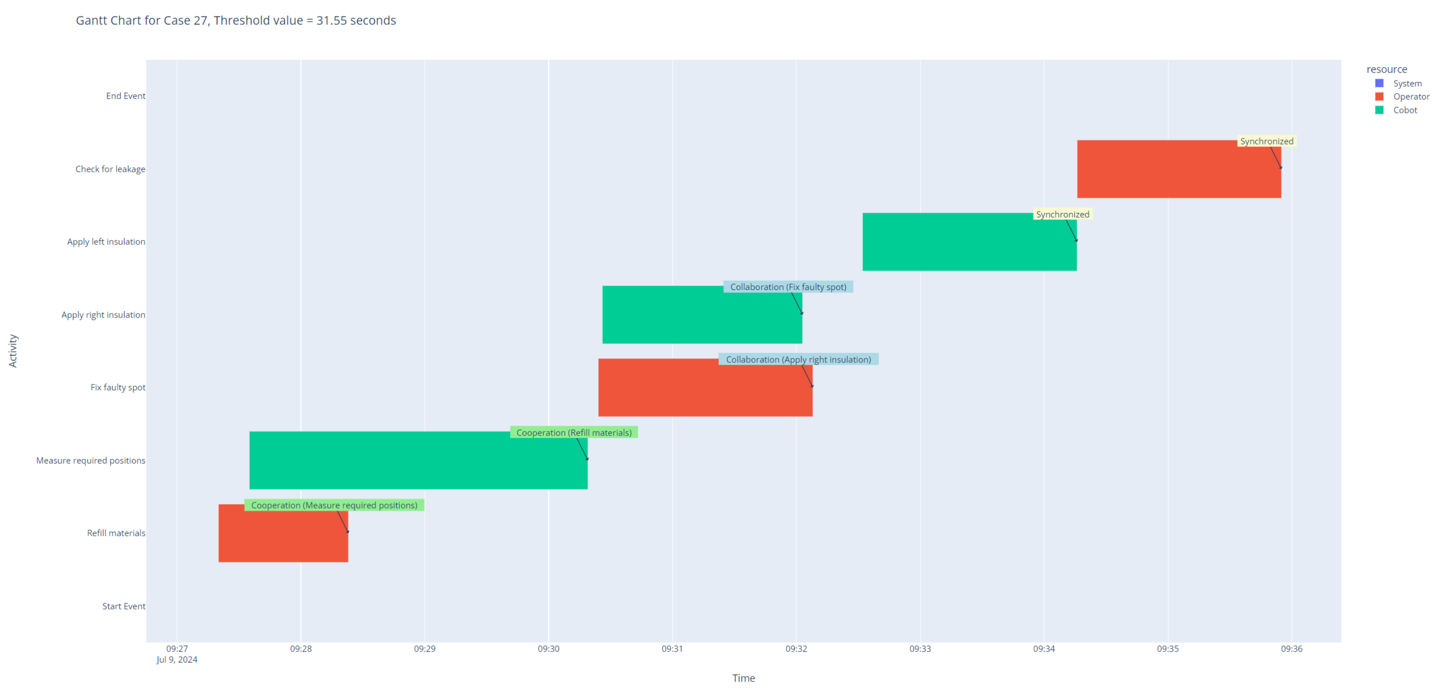


**Figure 9.9:** Wrongly classified process variant for "Refrigerator assembly"

### 9.1.6 Evaluation for Aircraft Cabin Assembly

The "Aircraft Cabin Assembly" use case involves the production of aircraft cabin components. A human and a cobot work simultaneously on one and the same part and perform different tasks that do not restrict each other in terms of time. Only one single process variant exists in this use case. The correct classification for this use case is entirely "Cooperation" for each activity, as seen in figure 9.10. The correct classification was achieved in 68% of all cases, making this a clear result. "Collaboration" was incorrectly detected in other cases due to the threshold value being higher than required.

| Combination | Process | Percentage |
|---|---|---|
| 1 | Cooperation, Cooperation, Cooperation, Cooperation, Cooperation | 68.0% |
| 2 | Collaboration, Cooperation, Collaboration, Cooperation, Collaboration | 18.0% |
| 3 | Collaboration, Collaboration, Collaboration, Collaboration, Collaboration | 6.1% |
| 4 | Collaboration, Cooperation, Collaboration, Cooperation, Cooperation | 4.0% |
| 5 | Cooperation, Collaboration, Cooperation, Collaboration, Collaboration | 3.9% |

**Table 9.4:** Discovered pattern combinations for "Aircraft Cabin Assembly"

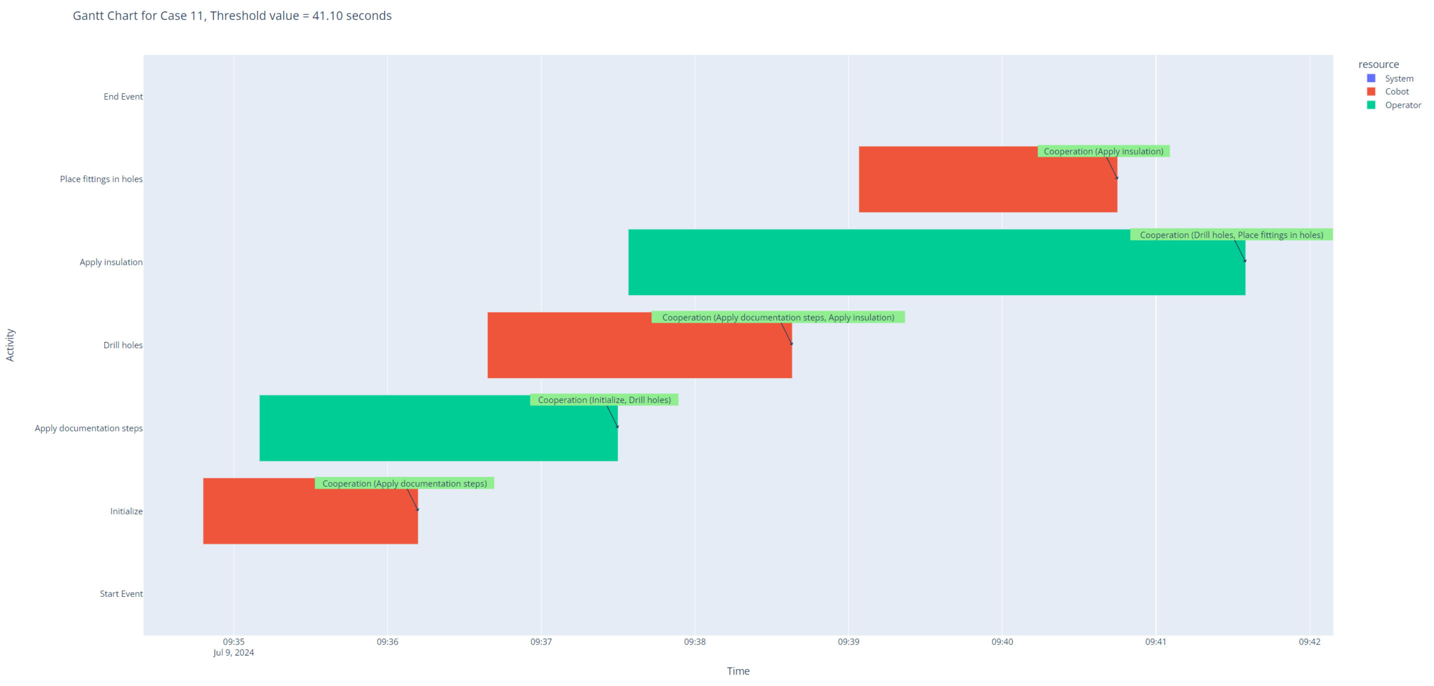


**Figure 9.10:** Correctly classified process variant for "Aircraft Cabin assembly"

Figure 9.11 shows a possible deviation from the correct classification: The activities "Initialize" and "Apply documentation steps", performed by the Operator, count towards a "Multiple overlapping" Collaboration with the activity "Apply documentation steps". This is this case because "Apply documentation steps" has a similar start timestamp as "Initialize" and a similar end timestamp as "Drill holes".

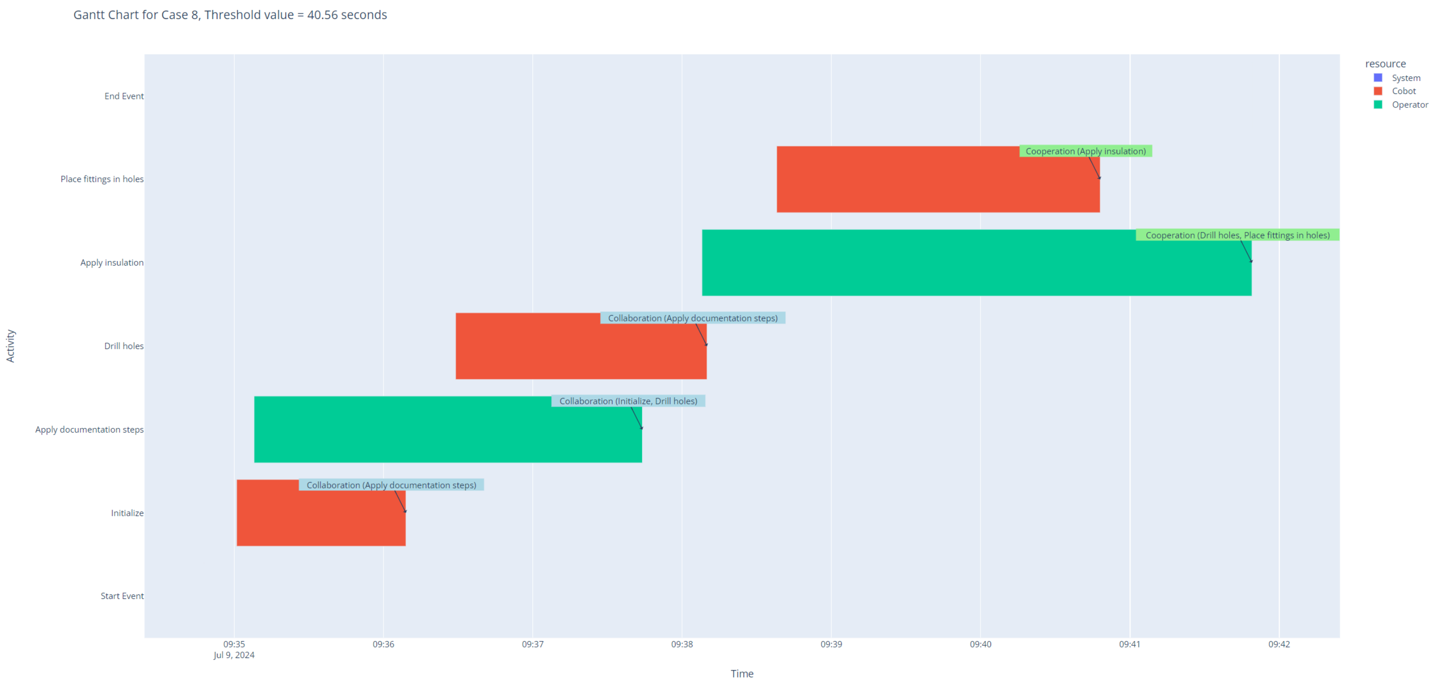


**Figure 9.11:** Wrongly classified process variant for "Aircraft Cabin assembly"

### 9.1.7 Evaluation for Box Assembly

"Box Assembly" is a use case that described the Cobot-assisted assembly of a wooden box through a human. It involved actual "Collaboration" work, meaning that the Cobot performs activities that support the human operator in achieving their goals. More precisely, the correct classification would be "Synchronized" for the activities "Start Robot", "Initialize", "Place first part", "Fit third part", "Fit fifth part" and "Move finished box away", while classifying the pairs of "Fit second part" / "Hold first part in place" as well as "Hold box in place" / "Fit fourth part" as "Collaboration". In the event log, only one process variant was discovered, however, 8 different combinations of discovered patterns were found, which are shown in table 9.5.

| **Combination** | **Process** | **Percentage** |
|---|---|---|
| 1 | Synchronized, Synchronized, Collaboration, Collaboration, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized | 73.5% |
| 2 | Synchronized, Synchronized, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 22.0% |
| 3 | Synchronized, Synchronized, Cooperation, Cooperation, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized | 2.2% |
| 4 | Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized | 1.0% |
| 5 | Synchronized, Synchronized, Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 0.8% |
| 6 | Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 0.2% |
| 7 | Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized | 0.2% |
| 8 | Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized | 0.1% |

**Table 9.5:** Discovered pattern combinations for "Box Assembly"

The correct classification was also by far the most common, accounting for 73.5% of all cases. Figure 9.12 shows the Gantt chart for the most common combination of patterns.

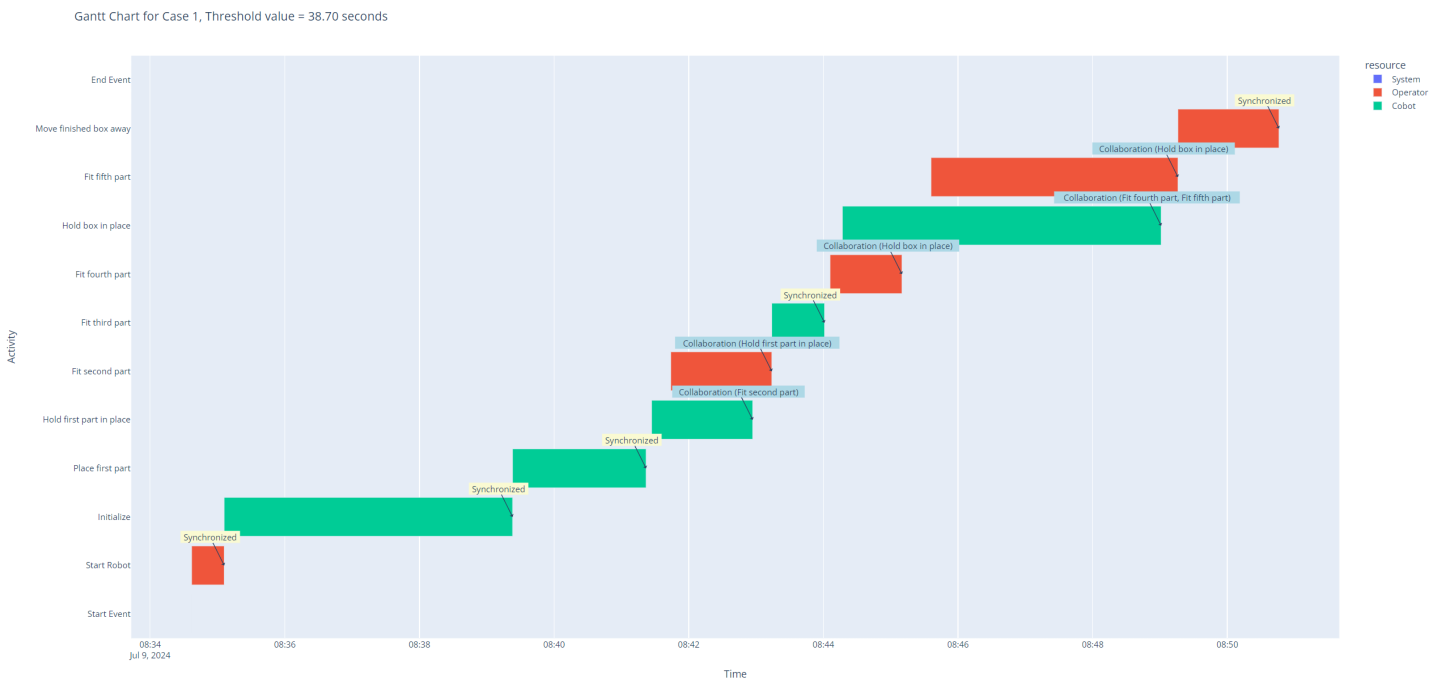


**Figure 9.12:** Correctly classified process variant for "Box assembly"

9.13 shows the most common wrong classification for "Box Assembly". It can be observed that the threshold value was too small so reliably identify the "Nearly complete Overlap" between "Hold box in place" and "Fit fourth part".

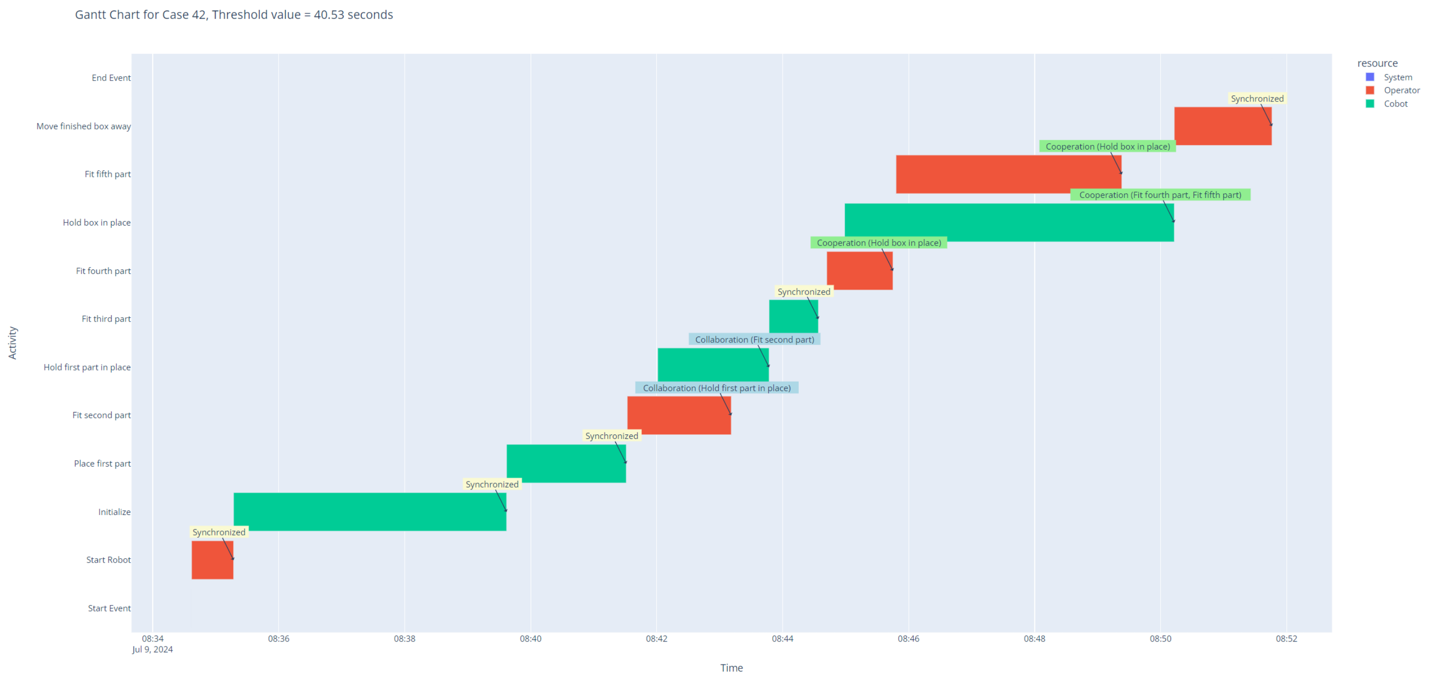


**Figure 9.13:** Wrongly classified process variant for "Box assembly"

### 9.1.8 Evaluation for Wheel Assembly

The last artificially generated event log "Wheel assembly" again describes a collaboration scenario between two actors or resources. The correct classification is "Synchronized" for all activities except the groups of "Hold left wheel assembly", "Fasten left screws" and "Guide left wheel assembly" as well as "Hold right wheel assembly", "Fasten right screws" and "Guide right wheel assembly", which are all "Collaboration". No "Cooperation" is intended in this scenario. This goal was achieved in 65.9% of all cases, which once again marks a clear result that does not leave room for any speculation. Only one process variant exists. Table 9.6 shows the 4 different combinations of discovered patterns.

| Combination | Process | Percentage |
|---|---|---|
| 1 | Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized | 65.9% |
| 2 | Synchronized, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized | 16.2% |
| 3 | Synchronized, Collaboration, Collaboration, Collaboration, Synchronized, Synchronized, Cooperation, Cooperation, Cooperation, Synchronized | 15.2% |
| 4 | Synchronized, Cooperation, Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation, Cooperation, Synchronized | 2.7% |

**Table 9.6:** Discovered pattern combinations for "Wheel Assembly"

Figure 9.14 displays a correctly classified case from the "Wheel Assembly" event log.

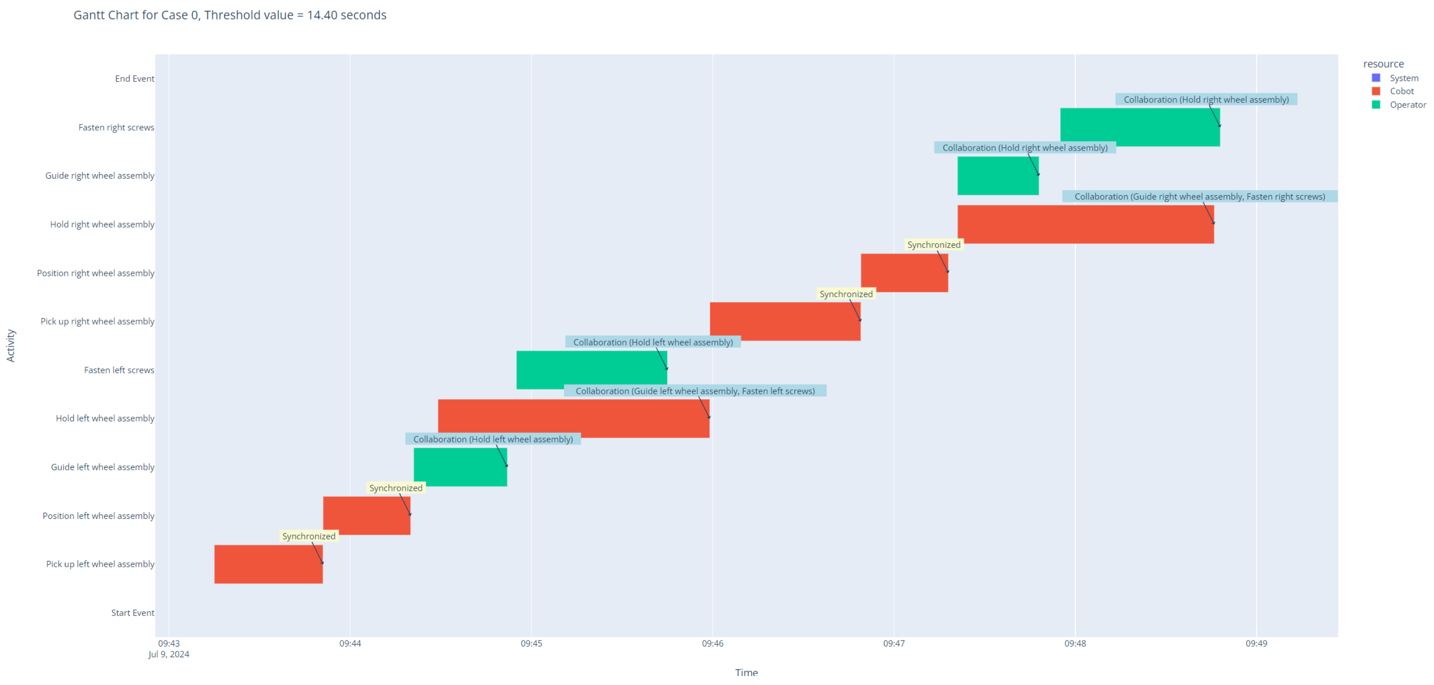


**Figure 9.14:** Correctly classified process variant for "Wheel assembly"

9.15 shows a common wrong classification for "Wheel Assembly". Due to a threshold that is too long, the "Multiple Overlapping" between the longer activity "Hold left wheel assembly" and the shorter activities "Guide left wheel assembly" and "Fasten left screws" was not detected, which means that those activities were classified as "Cooperation".

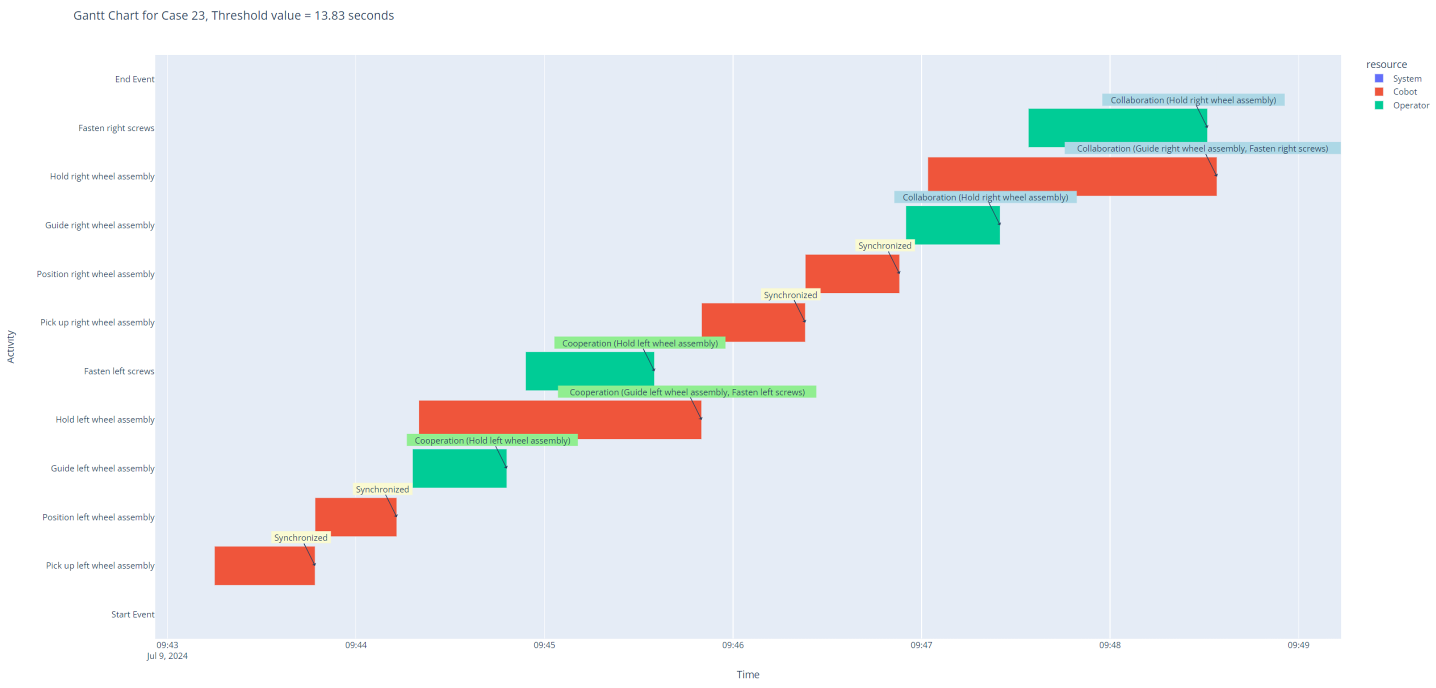
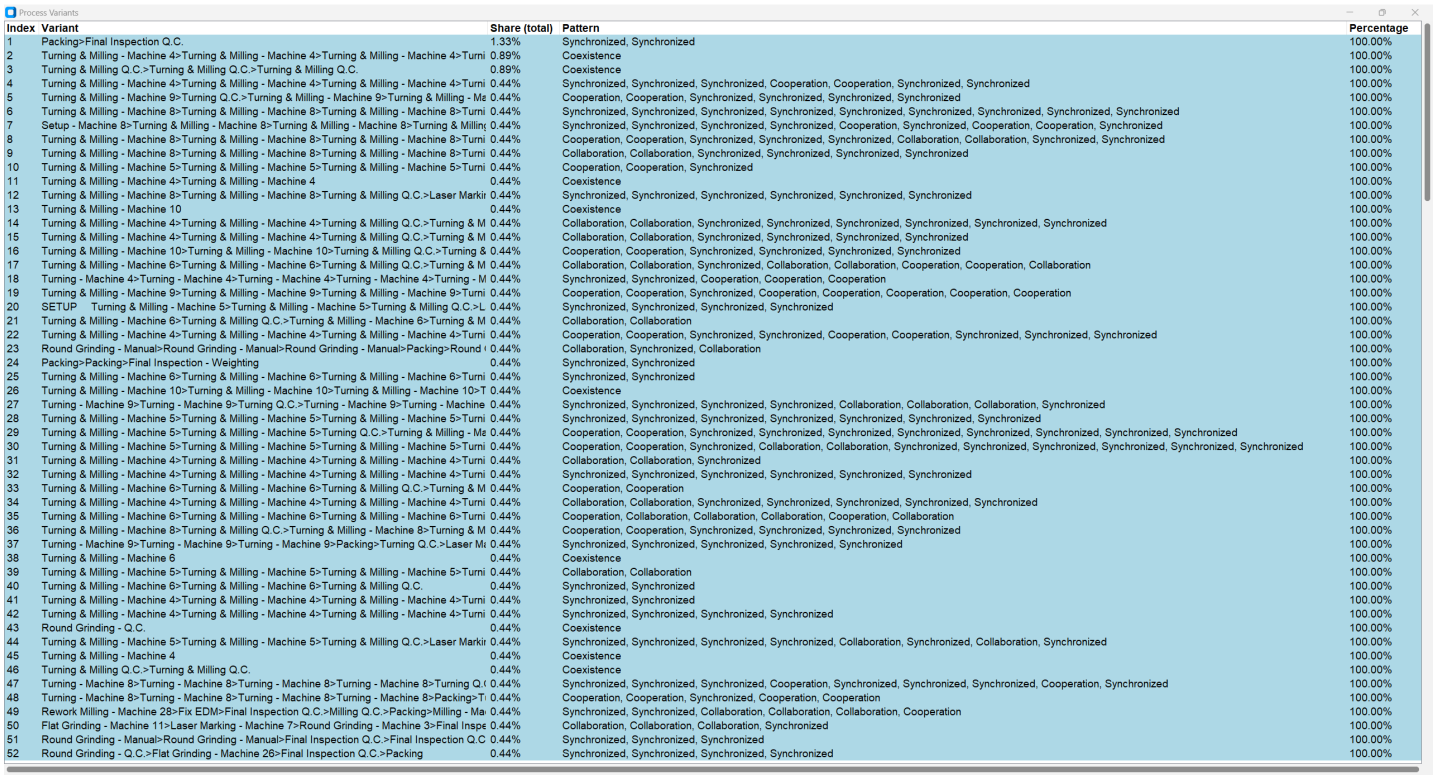


**Figure 9.15:** Wrongly classified process variant for "Wheel assembly"

### 9.1.9 Evaluation for Processing and Quality Inspection of Components

The event log "Processing and quality inspection of components" is unique compared to the other event logs that were used so far to evaluate the algorithm in this chapter: It was not generated using the event log generator as described in 5.2, but taken from an online source. More detail about the dataset are described in 5.1.9. As the dataset was not created as part of this work, the target patterns are unknown, however, the program was still used on the event log. The first notable detail is that other than the other event log, there are a total of 221 process variants, which means that the event log describes not just a simple process, but a large number of workflows within a company. For each process variant, the algorithm found exactly one combination of patterns, which means that the classification was always unambiguous.


Process Variants

| Index | Variant | Share (total) | Pattern | Percentage |
|---|---|---|---|---|
| 1 | Packing>Final Inspection Q.C. | 1.33% | Synchronized, Synchronized | 100.00% |
| 2 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.89% | Coexistence | 100.00% |
| 3 | Turning & Milling Q.C.>Turning & Milling Q.C.>Turning & Milling Q.C. | 0.89% | Coexistence | 100.00% |
| 4 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Synchronized, Synchronized, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized | 100.00% |
| 5 | Turning & Milling - Machine 9>Turning Q.C.>Turning & Milling - Machine 9>Turning & Milling - Ma | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 6 | Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turni | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 7 | Setup - Machine 8>Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turning & Milling | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Cooperation, Synchronized, Cooperation, Cooperation, Synchronized | 100.00% |
| 8 | Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turni | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized, Collaboration, Collaboration, Synchronized, Synchronized | 100.00% |
| 9 | Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turni | 0.44% | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 10 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turni | 0.44% | Cooperation, Cooperation, Synchronized | 100.00% |
| 11 | Turning & Milling - Machine 4>Turning & Milling - Machine 4 | 0.44% | Coexistence | 100.00% |
| 12 | Turning & Milling - Machine 8>Turning & Milling - Machine 8>Turning & Milling Q.C.>Laser Markir | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 13 | Turning & Milling - Machine 10 | 0.44% | Coexistence | 100.00% |
| 14 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling Q.C.>Turning & M | 0.44% | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 15 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling Q.C.>Turning & M | 0.44% | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 16 | Turning & Milling - Machine 10>Turning & Milling - Machine 10>Turning & Milling Q.C.>Turning & | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 17 | Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turning & Milling Q.C.>Turning & M | 0.44% | Collaboration, Collaboration, Synchronized, Collaboration, Collaboration, Cooperation, Cooperation, Collaboration | 100.00% |
| 18 | Turning - Machine 4>Turning - Machine 4>Turning - Machine 4>Turning - Machine 4>Turning - M | 0.44% | Synchronized, Synchronized, Cooperation, Cooperation, Cooperation | 100.00% |
| 19 | Turning & Milling - Machine 9>Turning & Milling - Machine 9>Turning & Milling - Machine 9>Turni | 0.44% | Cooperation, Cooperation, Synchronized, Cooperation, Cooperation, Cooperation, Cooperation, Cooperation | 100.00% |
| 20 | SETUP Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling Q.C.>L | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 21 | Turning & Milling - Machine 6>Turning & Milling Q.C.>Turning & Milling - Machine 6>Turning & M | 0.44% | Collaboration, Collaboration | 100.00% |
| 22 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Cooperation, Cooperation, Synchronized, Synchronized, Synchronized | 100.00% |
| 23 | Round Grinding - Manual>Round Grinding - Manual>Round Grinding - Manual>Packing>Round ( | 0.44% | Collaboration, Synchronized, Collaboration | 100.00% |
| 24 | Packing>Packing>Final Inspection - Weighting | 0.44% | Synchronized, Synchronized | 100.00% |
| 25 | Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turni | 0.44% | Synchronized, Synchronized | 100.00% |
| 26 | Turning & Milling - Machine 10>Turning & Milling - Machine 10>Turning & Milling - Machine 10>T | 0.44% | Coexistence | 100.00% |
| 27 | Turning - Machine 9>Turning - Machine 9>Turning Q.C.>Turning - Machine 9>Turning - Machine | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Synchronized | 100.00% |
| 28 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turni | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 29 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning Q.C.>Turning & Milling - Ma | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 30 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turni | 0.44% | Cooperation, Cooperation, Synchronized, Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 31 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Collaboration, Collaboration, Synchronized | 100.00% |
| 32 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 33 | Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turning & Milling Q.C.>Turning & M | 0.44% | Cooperation, Cooperation | 100.00% |
| 34 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Collaboration, Collaboration, Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 35 | Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turni | 0.44% | Cooperation, Collaboration, Collaboration, Collaboration, Cooperation, Collaboration | 100.00% |
| 36 | Turning & Milling - Machine 8>Turning & Milling Q.C.>Turning & Milling - Machine 8>Turning & M | 0.44% | Cooperation, Cooperation, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 37 | Turning - Machine 9>Turning - Machine 9>Turning - Machine 9>Packing>Turning Q.C.>Laser Ma | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 38 | Turning & Milling - Machine 6 | 0.44% | Coexistence | 100.00% |
| 39 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turni | 0.44% | Collaboration, Collaboration | 100.00% |
| 40 | Turning & Milling - Machine 6>Turning & Milling - Machine 6>Turning & Milling Q.C. | 0.44% | Synchronized, Synchronized | 100.00% |
| 41 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Synchronized, Synchronized | 100.00% |
| 42 | Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turning & Milling - Machine 4>Turni | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |
| 43 | Round Grinding - Q.C. | 0.44% | Coexistence | 100.00% |
| 44 | Turning & Milling - Machine 5>Turning & Milling - Machine 5>Turning & Milling Q.C.>Laser Markir | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized, Collaboration, Synchronized, Collaboration, Synchronized | 100.00% |
| 45 | Turning & Milling - Machine 4 | 0.44% | Coexistence | 100.00% |
| 46 | Turning & Milling Q.C.>Turning & Milling Q.C. | 0.44% | Coexistence | 100.00% |
| 47 | Turning - Machine 8>Turning - Machine 8>Turning - Machine 8>Turning - Machine 8>Turning Q.( | 0.44% | Synchronized, Synchronized, Synchronized, Cooperation, Synchronized, Synchronized, Synchronized, Cooperation, Synchronized | 100.00% |
| 48 | Turning - Machine 8>Turning - Machine 8>Turning - Machine 8>Turning - Machine 8>Packing>Tu | 0.44% | Cooperation, Cooperation, Synchronized, Cooperation, Cooperation | 100.00% |
| 49 | Rework Milling - Machine 28>Fix EDM>Final Inspection Q.C.>Milling Q.C.>Packing>Milling - Ma | 0.44% | Synchronized, Synchronized, Collaboration, Collaboration, Collaboration, Cooperation | 100.00% |
| 50 | Flat Grinding - Machine 11>Laser Marking - Machine 7>Round Grinding - Machine 3>Final Inspe | 0.44% | Collaboration, Collaboration, Collaboration, Synchronized | 100.00% |
| 51 | Round Grinding - Manual>Round Grinding - Manual>Final Inspection Q.C.>Final Inspection Q.C | 0.44% | Synchronized, Synchronized, Synchronized | 100.00% |
| 52 | Round Grinding - Q.C.>Flat Grinding - Machine 26>Final Inspection Q.C.>Packing | 0.44% | Synchronized, Synchronized, Synchronized, Synchronized | 100.00% |

**Figure 9.16:** Process variants for "Processing and quality inspection of components"

Another unique property of this event log is that breaks within activities are present, meaning that the same activity starts at one timestamp and ends at a later timestamp, but then starts and ends again after a certain period of time. As described in 6.1, if an activity occurs several times under the same name within a case, it is counted as a single activity for the recognition of patterns. This assumption was correctly realized by the program, as can be seen in numerous cases, such as case 30, which is shown in figure 9.17. In this case, both activities have breaks present, but the middle occurrences of "Turing & Milling - Machine 5" are still counted as "Cooperation" instead of "Synchronized", which they would otherwise be.

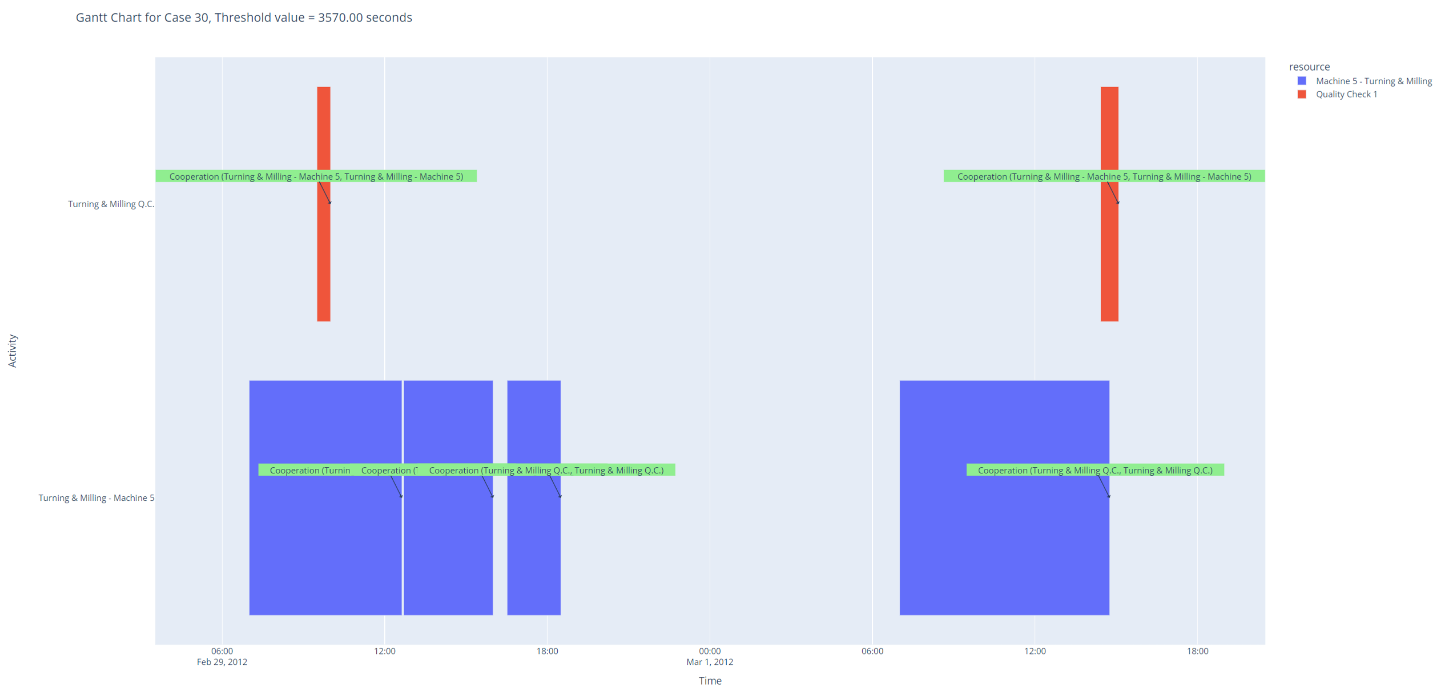


**Figure 9.17:** Case with several occurrences of the same activity in "Processing and quality inspection of components"

Due to the high number of process variants, no definitive answer about the patterns in this event log is possible. However, the program still helps in analyzing process variants present, which greatly facilitates further examination.

### 9.1.10 Summary of the Findings

After using the program on all event logs of the use cases described in section 5.1, the results were documented. Table 9.7 shows the percentage of correctly classified cases for each use case. It can be observed that event logs that only contain "Coexistence" and "Synchronized" patterns were always classified correctly in 100% of cases. This is not surprising, as these patterns are much easier to detect, while also being independent of the threshold value. Regarding the remaining use cases, the percentage of correct classifications ranges between 55.5% and 94.53%. Even when the number was only slightly above 50%, as in the case of the refrigerator assembly, the classification was always a clear result, meaning that process analysts could always make a clear assumption about the Collaborative Patterns present in the event logs.

| Use Case | Target Patterns | Percentage correctly classified | Unambiguous classification |
|---|---|---|---|
| Pick-and-Place | Coexistence | 100% | Yes |
| Hole Drilling | Coexistence | 100% | Yes |
| Glue Application | Synchronized | 100% | Yes |
| Spot Taping | Cooperation, Synchronized | 94.53% | Yes |
| Refrigerator Assembly | Cooperation, Synchronized | 55.5% | Yes |
| Aircraft Cabin Assembly | Cooperation | 68.0% | Yes |
| Box assembly | Collaboration, Synchronized | 73.5% | Yes |
| Wheel assembly | Collaboration, Synchronized | 65.9% | Yes |
| Processing and quality inspection of components | Various (Unknown) | Unknown | Yes |

**Table 9.7:** Results of the classification on the predefined cse cases

## 9.2 Threshold Value Analysis

Within the algorithm, the threshold value plays an important role, directly influencing the classification of the patterns of each activity. The threshold value does not affect the patterns "Coexistence" and "Synchronized", as these patterns are only assigned if there is no temporal overlap at all, ignoring the threshold value. In general, a higher threshold makes the classification of the "Collaboration" pattern more likely and the "Cooperation" pattern less likely. Consequently, a high threshold can mean that "Collaboration" is detected even in cases that should be "Cooperation" or even "Synchronized". This is due to the fact that shorter activities before/after a set of activities indicating "Collaboration" may be counted as "Collaboration" because they are completely within the threshold period. In turn, if the threshold is set too low, collaboration cases will not be detected correctly. Within the algorithm, the threshold value plays an important role, directly influencing the classification of the patterns of each activity. The threshold value does not affect the patterns "Coexistence" and "Synchronized", as these patterns are only assigned if there is no temporal overlap at all, ignoring the threshold value. In general, a higher threshold makes the classification of the "Collaboration" pattern more likely and the "Cooperation" pattern less likely. Consequently, a high threshold can mean that "Collaboration" is detected even in cases that should be "Cooperation" or even "Synchronized". This is due to the fact that shorter activities before/after a set of activities indicating "Collaboration" may be counted as "Collaboration" because they are completely within the threshold period. In turn, if the threshold is set too low, "Collaboration" cases will not be detected correctly.

### 9.2.1 Possible Options for the Threshold Value

There are two options for setting a threshold value, which means that either a static or a dynamic value is used. This subsection will describe the difference and which option was used.

**Setting the threshold value as a fixed time value**
In earlier iterations of the algorithm, the threshold value was set as a static number, which means a fixed number of seconds. This value is then used to determine patterns for the whole event log, not taking process variations into account, which bears the risk of using a non-fitting threshold value in a number of cases. Additionally, it makes analyzing the process beforehand necessary in order to select a fitting number, which is not possible in larger event logs.

**Using a dynamic value for the Threshold Value**
Due to the aforementioned disadvantages of using a static number, the decision was made to calculate a dynamic number for each case of the event log, directly influenced by the duration of the activities. This has several advantages, including:

- **No interference with other cases**: There is not a single threshold value for the whole event log, meaning that a fitting value is selected for each single case.
- **Possibility of analyzing event logs with many process variants**: With a large number of different process variants that differ greatly, the usage of a static number was not possible.

However, there is a risk that the algorithm will be problematic with highly fluctuating activity durations (high variance), making it sensitive to outliers in activity durations. There were two main options to use as a base number for calculating a threshold value

- **Mean duration of activities**: This option uses the mean duration of activities in order to calculate the threshold value for a case.
- **Median duration of activities**: This option uses the median duration of activities in order to calculate the threshold value for a case.

Table 9.8 compares the three options and their advantages and disadvantages.

| Threshold Type | Advantages | Disadvantages |
|---|---|---|
| **Fixed Time Value** | • Consistent basis for pattern detection across all cases.<br>• Simplifies calculation and implementation. | • Lack of adaptability to different cases.<br>• Context-insensitive, possibly leading to suboptimal detection. |
| **Mean Duration** | • Context-aware: adapts to all characteristics of the case.<br>• Mitigates minor variations in activity durations. | • Influenced by outliers, potentially skewing the threshold. |
| **Median Duration** | • Robust to outliers, providing stability in high variability.<br>• More representative central value for skewed distributions. | • Ignores distribution shape beyond the middle value.<br>• Less sensitive to bimodal or multimodal distributions. |

**Table 9.8:** Advantages and disadvantages of different threshold types for Collaboration Pattern detection

In production scenarios, multimodal distributions (distributions with more than one peak) are common [64], which means that utilizing the median might bear disadvantages in such scenarios. Thus, the mean duration was chosen for the algorithm because it offers a balanced approach, adapting to the characteristics of each case while effectively mitigating minor variations in activity durations, thereby ensuring a representative and context-aware threshold.

### 9.2.2 Selecting a fitting Percentage for the Threshold

Once the decision had been made to use the mean of the activity durations, a suitable percentage value had to be found that would enable correct classification in as many cases as possible and was therefore set as the default value in the program.
Table 9.9 compares different values for the threshold for each "Cooperation" and "Collaboration" use case. The analysis of "Coexistence" and "Synchronized" use cases was not necessary, as these are not influenced by the threshold value.

| Nominal pattern | Use Case | Value | Wrongly classified | Comment |
|---|---|---|---|---|
| Cooperation | Spottaping | 0% | 0% | Perfect detection if only cooperation is present. |
| Cooperation | Spottaping | 10% | 0% | Perfect detection if only cooperation is present. |
| Cooperation | Spottaping | 30% | 5.5% | Good detection. |
| Cooperation | Spottaping | 40% | 15.6% | Sufficient detection, but worse than 30%. |
| Cooperation | Refrigerator assembly | 0% | 0% | Perfect detection if only cooperation is present. |
| Cooperation | Refrigerator assembly | 10% | 9.9% | Good detection. |
| Cooperation | Refrigerator assembly | 30% | 45.2% | Worse result compared to 10%, however, still adequate. |
| Cooperation | Refrigerator assembly | 40% | 63.18% | Insufficient result. |
| Cooperation | Aircraft | 0% | 0% | With threshold of 0: Wrong detection of collaboration only possible with exactly the same timestamps of activities. |
| Cooperation | Aircraft | 10% | 1.5% | Good, almost perfect detection. |
| Cooperation | Aircraft | 30% | 32.1% | Sufficient detection. |
| Cooperation | Aircraft | 40% | 74% | Insufficient detection. |
| Collaboration | Box assembly | 0% | 100% | Threshold of 0 does not allow detection of collaboration. |
| Collaboration | Box assembly | 10% | 91.5% | Threshold too small for meaningful detection of collaboration. |
| Collaboration | Box assembly | 30% | 26.5% | Sufficient detection. |
| Collaboration | Box assembly | 40% | 38.8% | Sufficient detection, however, worse than 30%. |
| Collaboration | Wheel assembly | 0% | 100% | Threshold of 0 does not allow detection of collaboration. |
| Collaboration | Wheel assembly | 10% | 92.7% | Threshold that is too short does not recognise Collaboration with several parallel activities. |
| Collaboration | Wheel assembly | 30% | 34.1% | Sufficient detection. |
| Collaboration | Wheel assembly | 40% | 15.2% | Better detection than 30%. |

**Table 9.9:** Analysis of different threshold percentages

Generally, a detection can be considered reliable if less than 50% of cases were classified incorrectly. Thanks to the possibility of analyze process variants in the implementation, the user can easily analyze the frequency in which patterns were detected. In the use cases which were taken into consideration, this goal was achieved with a threshold set at 30% of the mean duration of activities. A value of 10% does not reliably detect Collaboration cases (e.g. "Box assembly" and "Wheel assembly") A value of 40% achieved worse performance than 30% in all but one case ("Wheel assembly"), which is why it was not taken into consideration. Generally speaking, a lower threshold value can be selected if the operator of the software knows that Collaboration is highly unlikely. In turn, if Collaboration is expected, a higher threshold value can be chosen.

## 9.3 Runtime Analysis

In this section, the runtime of the implementation will be estimated. In order to achieve this, the most computationally expensive parts of the progrtam will be analyzed, and their complexities combined. The common Big-O notation will be used.

### 9.3.1 Individual Program Parts

This subsection will describe the runtime of individual functions of the program. Generally, $n$ refers to the number of cases and $m$ to the number of activities within one case.

**Reading the Event Log ("read_event_log"):** This function reads a CSV file and converts it to a list of dictionaries. It has a linear complexity of $\mathcal{O}(n)$.

**Processing Each Case ("find_patterns_in_case"):** This function processes each case in order to identify patterns. To do so, it analyzes each existing pair of activities in a nested loop. It calls the functions needed to identify Collaboration Groups, Collaboration, Cooperation and Synchronized Cases. Each of these functions is linear, with the exception of checking for Collaboration groups, which iterates over the activities of the case once again in certain cases. Thus, for each case with $m$ events, the complexity of the nested loop within "find_patterns_in_case" plus the iteration over each activity within "is_collaboration_group" is $\mathcal{O}(m^3)$. For the total number of cases $n$, the complexity is quartic with $\sum_{i=1}^{n} \mathcal{O}(m_i^3)$.

**Displaying a process model using Inductive Miner ("display_petri_net"):** This function uses the Inductive Miner to create a process model for the event log. The worst-case runtime of the Inductive Mining algorithm is $\mathcal{O}(n^2)$ [65].

**Displaying a Gantt chart for the current case ("show_gantt_chart"):** This function creates and shows a Gantt Chart for the current case. As activities of unrelated cases do not need to be taken into consideration, the runtime for showing one chart is $\mathcal{O}(m)$.

**Saving the case list as a CSV file: ("save_as_csv"):** This method iterates over the existing list of cases in order to save them to a CSV file. For this reason, is also features a linear complexity of $\mathcal{O}(n)$.

### 9.3.2 Total Runtime for classifying all Cases of an Event Log

Combining the runtime of reading the Event log and processing each case, the following runtime is calculated:

$$\mathcal{O}(n) + \sum_{i=1}^{n} \mathcal{O}(m_i^3)$$

Due to the structure of the formula, the term $\sum_{i=1}^{n} \mathcal{O}(m_i^3)$ dominates. For this reason, the overall complexity can be simplified to:

$$\sum_{i=1}^{n} \mathcal{O}(m_i^3)$$

As an example calculation, a event log consisting of 1000 cases, which contain 5 activities each, the total complexity would be:

$$\begin{aligned}\sum_{i=1}^{1000} \mathcal{O}(5^3) &= 1000 \times 125 \\ &= 125000\end{aligned}$$

## 9.4 Challenges and Limitations

The following section refers to the challenges that occurred when recognizing Collaborative Patterns in event logs and the limitations of the algorithm. Please note that this section does not refer to the limitations of SLR, which can be found under 3.2.4.

### 9.4.1 Limited Information about Positioning of Activities

One challenge was the fact that regular event logs such as the ones being analyzed in this program only feature a limited number of information about the event log. The biggest shortcoming is that there is no information available that describes at which position of a workpiece an activity is carried out by the respective actor. In practice, this is generally possible, even for more complex processes, although additional hardware may be required for position detection (see section 9.6.2). When differentiating between the patterns "Cooperation" (working together on the same workpiece at the same time, but at different positions) and "Collaboration" (working together on the same workpiece at the same time, at the same position), position data would generally be required, but is not available. As long as no information about the exact positions is known, the algorithm can only make assumptions as to whether the "Cooperation" or "Collaboration" pattern is more likely based on time overlaps. The algorithm meets this challenge by defining precise cases in which the form of temporal overlap is most likely to be either "cooperation" or "collaboration". These can be found in section 6.2. A possible extension of the algorithm to tackle this challenge can be found in 9.6.2.

### 9.4.2 Quality of Detection dependent on Threshold Value

The "Threshold" value was introduced to enable high-quality pattern recognition in view of the individual event log. This can be used to influence whether "Cooperation" or "Collaboration" is recognized. The analysis of the correct value is described in section 9.2.2. However, this threshold value brings with it a new limitation, namely that the quality of the result can fluctuate with the threshold value. Based on table 9.9, different values are better suited to different use cases. In practice, however, it is not always known which value is most appropriate and users must either try out different values or rely on a value that is often suitable (e.g. 30%).

### 9.4.3 Only quantitative but no qualitative Assessment of the Patterns

The assessment of which pattern is actually present could still be incorrect, even based on the strict application of rules and the inclusion of temporal and spatial information. For example, this could happen if steps randomly always occur in a certain order, if humans and robots work in the same place but still independently of each other, or if the data quality is poor. In individual cases, a human could assess the result of the algorithm in order to verify whether the results are plausible, however, this is not possible for large event logs. One possible solution to this problem would be for the algorithm to also perform a qualitative analysis of the activities in the event log and thus decide which patterns are present. Section 9.6.1 discusses a possible extension of the algorithm to include this functionality.

## 9.5 Evaluation of the Implementation

The following section evaluates the implementation of the algorithm itself, in particularly whether the goals defined in section 7.1 have been met.

- **Ease of use**: A intuitive and modern-looking GUI was implemented using the CustomTkinter library in Python. The program is easy and intuitive to use, as buttons and program elements that are not required are only displayed when they are actually needed. If anything is unclear, the user can hover over elements or buttons of the program with the mouse to display tooltips.
- **Comprehensive options for analysis**: Once an event log has been opened in the program, the user can easily analyze individual cases by selecting them in the list and viewing the corresponding activities. Patterns are determined for each activity, and not just once for each case. Additionally, there is the possibility to display "Process variants" within the program, which allows even further analysis of individual process variants and their respective combinations of patterns.

- **High performance**: As described in 9.3, the program has a runtime of $\mathcal{O}(n) + \sum_{i=1}^{n} \mathcal{O}(m_i^3)$. Although this is a quartic runtime, it is limited to the number of activities (m) within a case, and not the larger number of total cases (n). Additionally, the implementation uses multi threading to calculate independent cases in parallel, saving time.
- **Graphical representation of cases**: The program is able to display individual cases as a Gantt chart, allowing the user to easily comprehend why the program made a particular classification.
- **Process discovery**: The program is able to create Process models (Petri Nets) using the inductive miner. Individual randomized colors were implemented for each actor so that the flow of work between actors can be traced even better.
- **High compatibility**: Due to being implemented in Python, the program can easily run on all major operating systems (Windows, Unix, OS X). Additional libraries need to be installed using the guide written in 7.2.

As evident in this list, the goals set in 7.1 have been met, however, the implementation still has some limitations:

- **Specific CSV format required**: The program assumes that the event log will always be in a specific CSV format, meaning that column names need to be exactly the once described in 7.3. If the naming is different, the program will not work correctly, meaning that the event log needs to be modified first.
- **Scalability**: The program achieves a high performance. Still, due to the presence of a nested loop within the program and the cubic runtime, the program might take a long processing time for very large event logs, especially if singular cases contain a large number of activities.

## 9.6 Future Improvements to the Program

This section presents possible enhancements that, if added to the implementation, could address the shortcomings mentioned in section 9.4.

### 9.6.1 Using Generative AI to identify Collaborative Patterns

Due to the nature of the current implementation, patterns are identified based on temporal overlapping and their respective Operator performing the task. There is however the possibility to use either a locally-hosted or cloud-based Large Language Model (Llm) in order to identify borderline cases, which could be either "Cooperation" or "Collaboration". The program would send a message to the Llm, which then answers with the most likely Collaborative pattern, which is then processed by the software. An exemplary prompt could look like the following:

```
Please answer in one word. There are 2 scenarios: Cooperation: Operator and
    robot perform separate activities on the same workpiece concurrently.
    Collaboration: Worker and robot work interactively towards the same
    goal on the same workpiece. Their actions depend on each other. The
    following activities are performed in parallel. Please tell me whether
    it is most likely "collaboration" or "cooperation". Robot: Hold
    Workpiece in place Human: Fasten screws
```

**Listing 9.1:** Possible prompt for Large language models

The AI would then answer accordingly, and the program could use the answer in its classification. During this work, a number of locally-hosted and cloud-based language models were tested. Table 9.10 shows the performance of each model tested.

| Name | Type | Quality of answer |
|---|---|---|
| GPT-4o [66] | Cloud-based | Promising results |
| GPT-3.5 [67] | Cloud-based | Understands task, but often answers wrong |
| Blenderbot-3B [68] | Local | Does not understand task |
| Gpt2-xl [69] | Local | Meaningless answer |
| Gpt-J-6B [70] | Local | Meaningless answer |
| Gpt-Neo-1.3B [71] | Local | Meaningless answer |
| Llama-7b-hf [72] | Local | Meaningless answer |

**Table 9.10:** Comparison of different LLms for identifying Collaborative Patterns

As shown in the table, only one Language model showed promising results, which was the paid GPT-4o model by OpenAI. All locally hosted and freely-available language models unfortunately performed badly. All models took a very long time to process single cases, which hinders practical usability. For this reason, the decision was made not to incorporate generative AI into the program at the current time. This might change in the future, once locally hosted Llm perform better and faster. The use of paid, cloud based language model would be possible, however, is was not implemented in order to keep the program free to use and not dependent on outside server and API availability.

### 9.6.2 Analyzing exact Positions on Workpieces

As described in 9.4.1, the quality of the recognition of the patterns "Cooperation" and "Collaboration" for the algorithm could be improved if additional information about the positions of the activities on the workpiece is available.
A possible event log incorporating such information could look like table 9.11:

| case_id | activity | start_timestamp | end_timestamp | resource | position |
|---|---|---|---|---|---|
| 0 | Start Event | 09.07.2024 09:27:14 | 09.07.2024 09:27:14 | System | |
| 0 | Measure required positions | 09.07.2024 09:27:34 | 09.07.2024 09:30:06 | Cobot | Full area |
| 0 | Refill materials | 09.07.2024 09:27:22 | 09.07.2024 09:28:34 | Operator | Not on workpiece |
| 0 | Fix faulty spot | 09.07.2024 09:30:16 | 09.07.2024 09:32:34 | Operator | Left section |
| 0 | Apply left insulation | 09.07.2024 09:30:24 | 09.07.2024 09:32:18 | Cobot | Left section |
| 0 | Apply right insulation | 09.07.2024 09:32:44 | 09.07.2024 09:35:20 | Cobot | Right section |
| 0 | Check for leakage | 09.07.2024 09:35:20 | 09.07.2024 09:37:27 | Operator | Left section |
| 0 | End Event | 09.07.2024 09:37:27 | 09.07.2024 09:37:27 | System | |

**Table 9.11:** Exemplary event log for the "Refrigerator Assembly" use case

If the information about the exact position of the work is available, it is much easier to differentiate between "cooperation" and "collaboration". For example, the "Cooperation" pattern would be assigned if the workpiece is being worked on at the same time but at different positions. In turn, the "Collaboration" pattern would be assigned if the position was the same. In practice, it is possible to recognize and collect such information. However, additional sensors would be required within the production facilities in order to record the exact positions.

# Chapter 10

# Conclusion

The aim of this thesis was to develop an algorithm to detect collaborative patterns from an event log, analyze the current state of research in the field of HRC and explore the challenges in detection. The findings are explained in the following chapter.
The work started with a SLR, in which an overview of the current state of research was gained. Literature on HRC in general and on collaborative patterns was identified and classified. A significant proportion of this literature consisted of use cases from industry, some of which were used in the further course of the work. Various collaboration scenarios were classified and discussed. The four most common patterns ("Coexistence", "Synchronized", "Cooperation" and "Collaboration") were identified as the current state of research, which is why they were used as the basis for recognition in the further course of the work. The research questions are answered in the following.

- **RQ1**: The question of how collaborative patterns can be automatically identified in an event log was answered by developing a suitable algorithm. This algorithm takes into account the temporal dependencies within the activities of a case in order to identify the most likely patterns. For this purpose, possible sequences and overlapping times between activities were visualized and the desired target pattern was defined for each case. The algorithm was implemented in Python, allowing it to be used on all common operating systems. An easy-to-use GUI has also been implemented for intuitive interaction. Furthermore, the implementation provides a range of additional options for analysis and visualization.

- **RQ2**: The biggest challenge in recognizing collaborative patterns is the limited amount of information contained in the event log. For example, there is usually no information about the positions of activities within the workpiece. However, this would make it easier to differentiate between the "Cooperation" and "Collaboration" patterns. The algorithm avoids this problem by analyzing which pattern is most likely to be present in the given form of temporal overlap.

By providing an overview of the current state of research in HRC and the development and implementation of an algorithm, this work can make an additional contribution to the progress of cooperation between humans and robots in production scenarios.

# Bibliography

[1] A. P. Andre Jungmittag, “The impacts of robots on labour productivity: A panel data approach covering 9 industries and 12 countries,” 2019. [Online]. Available: https://joint-research-centre.ec.europa.eu/document/download/6bb3305b-8608-4ed6-8a0d-2bc623240c12_en

[2] P. Barosz, G. Gołda, and A. Kampa, “Efficiency analysis of manufacturing line with industrial robots and human operators,” *Applied Sciences*, vol. 10, no. 8, 2020. [Online]. Available: https://www.mdpi.com/2076-3417/10/8/2862

[3] SICK Sensors, “Revolutionizing manufacturing: How robots boost productivity, quality, and safety.” [Online]. Available: https://www.therobotreport.com/revolutionizing-manufacturing-how-robots-boost-productivity-quality-and-safety/

[4] C. Edward, W. Wannasuphoprasit, and M. Peshkin, “Cobots: Robots for collaboration with human operators,” *Industrial Robot: An International Journal*, 03 1999.

[5] V. Villani, F. Pini, F. Leali, and C. Secchi, “Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications,” *Mechatronics*, vol. 55, pp. 248–266, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957415818300321

[6] KPMG, “The factory of the future: Industry 4.0 - the challenges of tomorrow.” [Online]. Available: https://assets.kpmg.com/content/dam/kpmg/es/pdf/2017/06/the-factory-of-the-future.pdf

[7] S. Samhaber and M. Leitner, “Collaborative patterns for workflows with collaborative robots,” in *Cooperative Information Systems*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2022, vol. 13591, pp. 131–148.

[8] C. Zhang Sprenger, J. A. Corrales Ramón, and N. U. Baier, “Rtmn 2.0—an extension of robot task modeling and notation (rtmn) focused on human–robot collaboration,” *Applied Sciences*, vol. 14, no. 1, 2024. [Online]. Available: https://www.mdpi.com/2076-3417/14/1/283

[9] O. Salunkhe, J. Stahre, D. Romero, D. Li, and B. Johansson, “Specifying task allocation in automotive wire harness assembly stations for human-robot

collaboration," *Computers & Industrial Engineering*, vol. 184, p. 109572, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S036083522300596X

[10] A. Dzedzickis, G. Vaičiūnas, K. Lapkauskaitė, D. Viržonis, and V. Bučinskas, "Recent advances in human–robot interaction: robophobia or synergy," *Journal of Intelligent Manufacturing*, 2024.

[11] B. Wilhelm, B. Manfred, M. Braun, P. Rally, and O. Scholtz, "Lightweight robots in manual assembly – best to start simply! examining companies' initial experiences with lightweight robots," 10 2016.

[12] R. Müller, M. Vette, and O. Mailahn, "Process-oriented task assignment for assembly processes with human-robot interaction," *Procedia CIRP*, vol. 44, pp. 210–215, 2016, 6th CIRP Conference on Assembly Technologies and Systems (CATS). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827116003620

[13] E. Matheson, R. Minto, E. G. G. Zampieri, M. Faccio, and G. Rosati, "Human–robot collaboration in manufacturing applications: A review," *Robotics*, vol. 8, no. 4, 2019. [Online]. Available: https://www.mdpi.com/2218-6581/8/4/100

[14] U. Othman and E. Yang, "An overview of human-robot collaboration in smart manufacturing," in *2022 27th International Conference on Automation and Computing (ICAC)*, 2022, pp. 1–6.

[15] A. Cesta, A. Orlandini, G. Bernardi, and A. Umbrico, "Towards a planning-based framework for symbiotic human-robot collaboration," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[16] T. Kopp, M. Baumgartner, and S. Kinkel, "Success factors for introducing industrial human-robot interaction in practice: an empirically driven framework," *The International Journal of Advanced Manufacturing Technology*, vol. 112, 01 2021.

[17] L. Wang, R. Gao, J. Váncza, J. Krüger, X. Wang, S. Makris, and G. Chryssolouris, "Symbiotic human-robot collaborative assembly," *CIRP Annals*, vol. 68, no. 2, pp. 701–726, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007850619301593

[18] E. Helms, R. Schraft, and M. Hagele, "rob@work: Robot assistant in industrial environments," in *Proceedings. 11th IEEE International Workshop on Robot and Human Interactive Communication*, 2002, pp. 399–404.

[19] I. Aaltonen, T. Salmi, and I. Marstio, "Refining levels of collaboration to support the design and evaluation of human-robot interaction in the

manufacturing industry," *Procedia CIRP*, vol. 72, pp. 93–98, 2018, 51st CIRP Conference on Manufacturing Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827118303743

[20] S. El Zaatari, M. Marei, W. Li, and Z. Usman, "Cobot programming for collaborative industrial tasks: An overview," *Robotics and Autonomous Systems*, vol. 116, pp. 162–180, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092188901830602X

[21] W. Aalst and A. Weijters, "Process mining: A research agenda," *Computers in Industry*, vol. 53, pp. 231–244, 06 2004.

[22] E. Iman, M. D. Laanaoui, and H. Sbai, "Applying process mining to sensor data in smart environment: A comparative study," in *Innovations in Smart Cities Applications Volume 6*, M. Ben Ahmed, A. A. Boudhir, D. Santos, R. Dionisio, and N. Benaya, Eds. Cham: Springer International Publishing, 2023, pp. 511–522.

[23] W. M. P. van der Aalst, *Process Mining: Data Science in Action*, 2nd ed. Heidelberg: Springer, 2016.

[24] A. Rozinat and W. Aalst, "Conformance checking of processes based on monitoring real behavior," *Information Systems*, vol. 33, pp. 64–95, 03 2008.

[25] J. Webster and R. Watson, "Analyzing the past to prepare for the future: Writing a literature review," *MIS Quarterly*, vol. 26, 06 2002.

[26] Y. Levy and T. Ellis, "A systems approach to conduct an effective literature review in support of information systems research," *International Journal of an Emerging Transdiscipline*, vol. 9, 01 2006.

[27] D. Bem, "Writing a review article for psychological bulletin," *Psychological Bulletin*, vol. 118, pp. 172–177, 09 1995.

[28] J. Krueger, T. K. Lien, and A. Verl, "Cooperation of human and machines in assembly lines," *CIRP Annals*, vol. 58, no. 2, pp. 628–646, 2009.

[29] J. Edward, W. Wannasuphoprasit, and M. Peshkin, "Cobots: Robots for collaboration with human operators," *Proceedings of the INternational Mechanical Engineering Congress and Exhibition*, 03 1999. [Online]. Available: https://peshkin.mech.northwestern.edu/publications/1996_Colgate_CobotsRobotsCollaboration.pdf

[30] S. Kumar, C. Savur, and F. Sahin, "Survey of human–robot collaboration in industrial settings: Awareness, intelligence, and compliance," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 1, pp. 280–297, 2021.

[31] J. Bauer, A. Amine, S. Kurscheid, F. Lucas, G. Lozenguez, and R. Daub, "Towards enabling human-robot collaboration in industry: Identification of current

implementation barriers," in *5th Conference on Production Systems and Logistics*, 2023.

[32] Y. Liu, G. Caldwell, M. Rittenbruch, M. Belek Fialho Teixeira, A. Burden, and M. Guertler, "What affects human decision making in human–robot collaboration?: A scoping review," *Robotics*, vol. 13, no. 2, 2024. [Online]. Available: https://www.mdpi.com/2218-6581/13/2/30

[33] L. Wang, "Current status and future trends on human-robot collaboration," in *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2022, pp. 3–3.

[34] V. Villani, F. Pini, F. Leali, and C. Secchi, "Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications," *Mechatronics*, vol. 55, pp. 248–266, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957415818300321

[35] U. Othman and E. Yang, "Human–robot collaborations in smart manufacturing environments: Review and outlook," *Sensors*, vol. 23, no. 12, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/12/5663

[36] T. Munzer, M. Toussaint, and M. Lopes, "Efficient behavior learning in human–robot collaboration," *Autonomous Robots*, vol. 42, no. 5, pp. 1103–1115, 2018.

[37] Nadine Winkelmann, "Human-robot cooperation at audi," 2017. [Online]. Available: https://www.springerprofessional.de/manufacturing/production---production-technology/human-robot-cooperation-at-audi/14221870

[38] A. Cherubini, R. Passama, P. Fraisse, and A. Crosnier, "A unified multimodal control framework for human–robot interaction," *Robotics and Autonomous Systems*, vol. 70, pp. 106–115, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889015000391

[39] S. Lichiardopol, N. van de Wouw, and H. Nijmeijer, "Control scheme for human-robot co-manipulation of uncertain, time-varying loads," in *2009 American Control Conference*, 2009, pp. 1485–1490.

[40] I. El Makrini, K. Merckaert, D. Lefeber, and B. Vanderborght, "Design of a collaborative architecture for human-robot assembly tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1624–1629.

[41] Universal Robots Academy, "An introduction to common collaborative robot applications," 2022. [Online]. Available: https://www.universal-robots.com/au/whitepaper-discover-common-cobot-applications/

[42] V. Román Ibáñez, F. Pujol, S. García Ortega, and J. Sanz Perpiñán, "Collaborative robotics in wire harnesses spot taping process," *Computers*

*in Industry*, vol. 125, p. 103370, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361520306047

[43] M. Fujii, H. Murakami, and M. Sonehara, "Study on application of a human-robot collaborative system using hand-guiding in a production line," *IHI Eng. Rev*, vol. 49, no. 1, pp. 24–29, 2016.

[44] G. Michalos, S. Makris, P. Tsarouchi, T. Guasch, D. Kontovrakis, and G. Chryssolouris, "Design considerations for safe human-robot collaborative workplaces," *Procedia CIRP*, vol. 37, pp. 248–253, 2015, cIRPe 2015 - Understanding the life cycle implications of manufacturing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827115008550

[45] Stefan Samhaber, "Collaboration patterns for collaborative robots," Master's thesis, Universität Wien, Wien, 2022.

[46] International Federation of Robotics, "Demystifying collaborative industrial robots," 2020. [Online]. Available: https://ifr.org/papers/demystifying-collaborative-industrial-robots-updated-version

[47] P. Segura, O. Lobato-Calleros, A. Ramírez-Serrano, and I. Soria, "Human-robot collaborative systems: Structural components for current manufacturing applications," *Advances in Industrial and Manufacturing Engineering*, vol. 3, p. 100060, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666912921000301

[48] A. Schäfer, S. Kinkel, and T. Kopp, "Kollaborierende oder kollaborationsfähige roboter?" *Industrie 4.0 Management*, vol. 36, no. 2, pp. 19–23, 2020. [Online]. Available: https://doi.org/10.30844/I40M_20-2_S19-23

[49] A. Baratta, A. Cimino, F. Longo, G. Mirabelli, and L. Nicoletti, "Task allocation in human-robot collaboration: A simulation-based approach to optimize operator's productivity and ergonomics," *Procedia Computer Science*, vol. 232, pp. 688–697, 2024, 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050924000681

[50] A. Castro, F. Silva, and V. Santos, "Trends of human-robot collaboration in industry contexts: Handover, learning, and metrics," *Sensors*, vol. 21, no. 12, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/12/4113

[51] S. Haddadin and E. Croft, *Physical Human–Robot Interaction*. Cham: Springer International Publishing, 2016, pp. 1835–1874. [Online]. Available: https://doi.org/10.1007/978-3-319-32552-1_69

[52] X. V. Wang, Z. Kemény, J. Váncza, and L. Wang, "Human–robot collaborative assembly in cyber-physical production: Classification framework and

implementation," *CIRP Annals*, vol. 66, no. 1, pp. 5–8, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0007850617301014

[53] J. Lonchamp, "Process model patterns for collaborative work," in *15th IFIP World Computer Congress - Telecooperation'98*, Vienna, Austria, Aug. 1998, p. 12 p, colloque avec actes et comité de lecture. [Online]. Available: https://inria.hal.science/inria-00107838

[54] Y. Verginadis, N. Papageorgiou, D. Apostolou, and G. Mentzas, "A review of patterns in collaborative work," in *Proceedings of the 2010 ACM International Conference on Supporting Group Work*, ser. GROUP '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 283–292. [Online]. Available: https://doi.org/10.1145/1880071.1880118

[55] T. T. Vo, B. Coulette, H. N. Tran, and R. Lbath, "An approach to define and apply collaboration process patterns for software development," in *Model-Driven Engineering and Software Development*, P. Desfray, J. Filipe, S. Hammoudi, and L. F. Pires, Eds. Cham: Springer International Publishing, 2015, pp. 248–262.

[56] M. L. Cisse, H. N. Tran, S. Diaw, B. Coulette, and A. Bah, "Using patterns to parameterize the execution of collaborative tasks," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2019, pp. 106–111.

[57] D. Yu and J. Wang, "Mining collaboration patterns of software development processes based on trace alignment," in *Proceedings of the 2017 International Conference on Software and System Process*, ser. ICSSP 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 15–24. [Online]. Available: https://doi.org/10.1145/3084100.3084103

[58] F. Corradini, S. Pettinari, B. Re, L. Rossi, and F. Tiezzi, "A technique for discovering bpmn collaboration diagrams," *Software and Systems Modeling*, 2024.

[59] T. T. Vo, B. Coulette, H. N. Tran, and R. Lbath, "Defining and using collaboration patterns for software process development," in *2015 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015, pp. 557–564.

[60] Wired Workers, "Cobot end of arm tooling." [Online]. Available: https://www.wiredworkers.io/cobot/tools/end-of-arm-tooling/

[61] D. Levy, "Production analysis with process mining technology," 2014. [Online]. Available: https://data.4tu.nl/articles/_/12697997/1

[62] A. Berti, S. van Zelst, and D. Schuster, "Pm4py: A process mining library for python," *Software Impacts*, vol. 17, p. 100556, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963823000933

[63] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, “Discovering block-structured process models from event logs - a constructive approach,” in *Application and Theory of Petri Nets and Concurrency*, J.-M. Colom and J. Desel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 311–329.

[64] R. C. H. Cheng and C. S. M. Currie, “Input modelling for multimodal data,” *Journal of the Operational Research Society*, vol. 71, no. 6, pp. 1038–1052, 2020. [Online]. Available: https://doi.org/10.1080/01605682.2019.1609887

[65] W. M. P. van der Aalst, *Foundations of Process Discovery*. Cham: Springer International Publishing, 2022, pp. 37–75. [Online]. Available: https://doi.org/10.1007/978-3-031-08848-3_2

[66] OpenAI, “Gpt-4 technical report,” 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[67] ——, “Language models are few-shot learners,” 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2005.14165

[68] MetaAI, “Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage,” 2022. [Online]. Available: https://arxiv.org/abs/2208.03188

[69] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[70] B. Wang and A. Komatsuzaki, “GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model,” https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[71] S. Black, L. Gao, P. Wang, C. Leahy, and S. Biderman, “Gpt-neo: Large scale autoregressive language modeling with mesh-tensorflow,” 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:245758737

[72] MetaAI, “Llama: Open and efficient foundation language models,” 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

## Erklärung an Eides statt

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher an keiner anderen Hochschule zur Erlangung eines akademischen Grades eingereicht. Die vorgelegten Druckexemplare und die dem Prüfer/der Prüferin zur Verfügung gestellte elektronische Version (PDF-Datei) der Arbeit sind identisch. Von den in §13 Abs. 3 PO 2015 vorgesehenen Rechtsfolgen habe ich Kenntnis.

Regensburg, den 09.09.2024

Benjamin Meier
Matrikelnummer 2326517