MASTER

Process mining for systems with automated batching
an exploratory study on new process mining grounds

Verhaegh, K.W.

Award date:
2018
Process Mining for Systems with Automated Batching

An Exploratory Study on New Process Mining Grounds

By:
K.W. (Koen) Verhaegh
Student Number: 0860736

Supervisors:
Dr. D. Fahland, TU/e
Dr. R. Eshuis, TU/e
E. Blokhuis, Vanderlande Industries B.V.
H. F. Bernard, Vanderlande Industries B.V.

Veghel, August 2018
ABSTRACT

Process mining techniques have been developed for and applied on multiple domains: information handling systems (IHS) with single case identifiers, IHSs with multiple case identifiers, and material handling systems (MHS) with single case identifiers. However, process mining has not been researched for MHSs with multiple case identifiers. The goal of this research is to identify the applicability and limitations of current existing process mining techniques for this specific environment, and to define a methodology to do successful analysis. Therefore, in this thesis, an exploratory study is presented, which analyzes multiple approaches and tools. Following CRISP-DM, the research presents a conceptualization of warehouse automation systems (a MHS with multiple case identifiers) and presents abstract process models that help with setting up log extraction. A semi-automatic log extraction approach is presented that first extracts a Log Extraction Specification and then extracts an event log. The Log Extraction Specification specifies the lifecycle of each object in the system and serves as an input for the log extraction algorithm. This two-step approach is used to extract logs from the case study system. Various proposed pre-processing approaches and tools are evaluated on multiple logs in a combinatorial study. The defined methodology is presented that summarizes the findings and explains how various process mining techniques can be applied on (other) MHSs with multiple case identifiers. The thesis is concluded by highlighting multiple future research problems to further investigate this domain of process mining.

Keywords: Process mining, material handling systems conceptualization, lifecycle models, log extraction specification, event log extraction and pre-processing, multiple case identifiers
PREFACE

This master thesis is the concluding chapter of my 5-year study in both Computer Science (bachelor) and Business Information Systems (master) at the Eindhoven University of Technology. It is the result of my master graduation project, done in cooperation with Vanderlande Industries B.V. and the Architecture of Information Systems group of the Mathematics and Computer Science department of the university.

First and foremost, I would like to thank my graduation supervisor Dirk Fahland for his guidance during the project and as a mentor during the master’s program. I highly appreciate the interesting discussions, brainstorming sessions, reviews, and constructive feedback throughout the entire project. I would also like to thank Rik Eshuis for his efforts in being my second assessor.

Furthermore, I would like to thank everyone at Vanderlande Industries B.V. in Veghel. I would like to thank them for giving me the opportunity to work with experts in the field of automation and for making me feel welcomed and at home. Everyone I have met at Vanderlande has been very positive and open and has helped me develop as a person and professionally. Special thanks go out to Erik Blokhuis and Hilda Bernard, my supervisors, who helped me get going in the company and guided me throughout the entire internship. Moreover, I would also like to specifically thank Martijn Sondermeijer and Maresa Bunschoten, who helped me greatly with obtaining data for analysis and the evaluation of the results. Lastly, I would also like to thank Vadim Denisov for his previous work on material handling systems at Vanderlande and his insights for my project. I am grateful to have worked with all of you!

Finally, I would like to thank my family, friends, girlfriend, and newly made friends at both the university and Vanderlande for supporting me during this project, and for some for many years. Thank you to everyone that has helped me reach this point in my personal and professional life.
TABLE OF CONTENTS

ABSTRACT .............................................................................................................................. 2

PREFACE .................................................................................................................................. 3

LIST OF FIGURES .................................................................................................................. 6

1. INTRODUCTION .............................................................................................................. 8
  1.1. Motivation and Context ............................................................................................. 8
  1.2. Research Problem and Methodology ...................................................................... 10
  1.3. Case Study ................................................................................................................ 11
  1.4. Approach and Results ............................................................................................ 12

2. PRELIMINARIES ............................................................................................................. 14
  2.1. Process Mining and Business Processes ................................................................. 14
  2.2. Multiple Case Identifiers and Batching .................................................................. 17
  2.3. Material Handling Systems .................................................................................... 19

3. CONCEPTUAL MODEL OF WAREHOUSE AUTOMATION ........................................... 21
  3.1. Warehouse Processes ............................................................................................. 21
  3.2. Warehouse Automation System .............................................................................. 24

4. LIFECYCLE MODELS .................................................................................................... 29
  4.1. Modules vs. Complete Process .............................................................................. 29
  4.2. Lifecycle Model Concepts ....................................................................................... 29
  4.3. System Lifecycle Model of a Tray ......................................................................... 32
  4.4. System Lifecycle Models of a Pallet ...................................................................... 33
  4.5. Complete System Lifecycle Model ......................................................................... 35
  4.6. Concluding the Conceptualization of Warehouse Automation Systems .................. 38

5. EVENT LOG EXTRACTION ............................................................................................ 39
  5.1. Requirements for Log Extraction ........................................................................... 40
  5.1.1. Source Data ......................................................................................................... 40
  5.1.2. Specifying a Case Identifier ............................................................................... 41
  5.1.3. Scoping and Case Time-frames ......................................................................... 42
  5.1.4. Log Extraction Specification ............................................................................. 43
  5.2. Extracting Log Extraction Specifications .................................................................. 44
  5.3. Extracting Event Logs ............................................................................................. 47
  5.3.1. Event Log Extraction with Default View .......................................................... 47
  5.3.2. Event Log Extraction with Other Views ............................................................ 48
  5.4. Extracted Event Logs ............................................................................................... 49

6. EVENT LOG PRE-PROCESSING .................................................................................... 50
  6.1. Quality Criteria ........................................................................................................ 50
  6.2. Data Exploration ....................................................................................................... 51
LIST OF FIGURES

Figure 1: Four quadrants of systems and case identifiers ......................................................... 9
Figure 2: Cross-industry standard process for data mining (CRISP-DM) model and the slightly adjusted version ................................................................. 11
Figure 3: Process Mining (red) and log extraction (green), as described in the Process Mining Manifesto [1] ................................................................. 14
Figure 4: Event log example as a table in Excel, sorted on time (left), and sorted on CaseID (right) ...................... 15
Figure 5: Event log example in CSV format (left), and other representations of event logs (right) .......... 15
Figure 6: Disco, as taken from Fluxicon's website .................................................................... 16
Figure 7: A performance spectrum, as discovered in the Performance Spectrum Miner .................. 19
Figure 8: An overview of the process in an automated warehouse ............................................. 21
Figure 9: Toy example of filling 2 trays from 2 pallets and stacking 2 orders, annotated with the process steps from Figure 8 ........................................ 23
Figure 10: Toy example warehouse blueprint; edges indicate what object is being moved .................... 24
Figure 11: All communication with the modules goes only to and from the Conductor ...................... 25
Figure 12: High level overview of the multiple tables in the database ......................................... 25
Figure 13: Example events in two different tables .................................................................... 26
Figure 14: Simplified MSD for the described conceptual WAS (with process step numbers) .............. 26
Figure 15: Simplified MSD showing the 1:n relation between a pallet and trays (with process step numbers) ........................................................................ 28
Figure 16: System lifecycle model of a tray, annotated with the process steps of Figure 8 .................. 31
Figure 17: Inbound (left) and outbound (right) pallet system lifecycle model, annotated with the process steps of Figure 8 .............................................. 34
Figure 18: Example concurrency between pallet and tray events ................................................. 36
Figure 19: Complete system lifecycle model for pallet & tray combined, annotated with process steps from Figure 8 ........................................................................ 37
Figure 20: Pipeline approach for extracting event logs, by first defining and extracting a log extraction specification ........................................................................... 40
Figure 21: Data examples for start and end events for the running example to the left ....................... 41
Figure 22: Running example: partial order for 2 pallets, 2 trays, and 2 shop orders ......................... 41
Figure 23: The Log Extraction Specification per case .................................................................... 43
Figure 24: Obtained LES for running example (showing 4 cases, 1 per row) ................................. 44
Figure 25: Event log of running example, for case P1-Ta-S01 ...................................................... 44
Figure 26: Extracting the LES from various tables ..................................................................... 45
Figure 27: Joining events and removing the wrong ones to obtain the correct case timeframes ............ 46
Figure 28: Model of the full process in Disco ............................................................................. 51
Figure 29: Model of the full process, with an aggregated event log ................................................ 52
Figure 30: (Aggregated) Physical flow model of the complete process; 100% (left) vs 0% (right) ........................................................................................................ 53
Figure 31: Renaming various storage locations to one general 'stored' activity to reduce model size .......... 54
Figure 32: Cutout of full model, showing a nice pattern .................................................................. 59
Figure 33: Inbound pallet model (projection) ............................................................................. 60
Figure 34: High level aggregation, showing the process on a high abstraction level ......................... 61
Figure 35: Comparison of 3 stacker module instances .................................................................. 61
Figure 36: C-net model on the physical flow log, discovered with the Interactive Data-aware Heuristics Miner ......................................................................................... 62
Figure 37: Part of the model discovered by the inductive miner, illustrating the many parallel paths .................................................................................................................. 63
Figure 38: Excerpt of another model discovered by the inductive miner, illustrating a flower model .................................................................................................................. 63
Figure 39: Initial performance spectrum discovered by the PSM (after manual ordering) ................. 63
Figure 40: Performance spectrum showing only the traces .......................................................... 64
Figure 41: Performance spectrum pointing at a recurring pattern that seems to be problematic, but is not .................................................................................................................. 64
Figure 42: Physical flow model with a 1:n:n case id, at 20% paths .................................................. 65
Figure 43: BPMN (left) and Petri net (right) model of de-stacking .................................................... 79
Figure 44: BPMN (left) and Petri net (right) model of stacking ....................................................... 80
Figure 45: BPMN model for retrieving, storing, and transporting .................................................... 80
Figure 46: Full model, with context added through numbering .......................................................... 81
Figure 47: Full model, with context added through specific information, and a circle highlighting the cutout of Figure 47 ..................................................................................................... 82
Figure 48: Cutout of Figure 47, showing a nice pattern .................................................................... 82
Figure 49: "High level" order model, 0% paths (left) and 100% paths (right) ...................................... 83
Figure 50: Order model with added context through numbering, 0% paths (left), 100% paths (right) ..................................................................................................................... 84
Figure 51: Order model with specific context added (destination) .................................................................85
Figure 52: Inbound pallet model (projection) ...................................................................................................86
Figure 53: De-stacking model of pallet and tray (projection) ..........................................................................87
Figure 54: De-stacking to stacking model of tray only (projection) .................................................................88
Figure 55: High level aggregation, showing the process on a high abstraction level .......................................89
Figure 56: High level aggregation, showing the model on a slightly lower level of abstraction .......................90
Figure 57: Comparison of 3 stacker module instances ....................................................................................91
Figure 58: Excerpt of the comparison of 3 stacker module instances, in performance view .............................91
Figure 59: Animation on the model also clearly shows the delay (when playing) .............................................92
Figure 60: Model of split log of tray before storage, in performance view .....................................................93
1. **INTRODUCTION**

This master thesis is the result of a graduation project conducted at Vanderlande Industries B.V. as part of the Business Information Systems master at Eindhoven University of Technology, in the Architecture of Information Systems (AIS) research group.

In this chapter the motivation and context for this research is given in Section 1.1. This builds up to the research problem and chosen methodology as described in Section 1.2. In both these sections the current state of process mining is considered and shortly evaluated for the purpose of application in material handling systems. We start with a general look into the process mining landscape, highlighting where the challenges for this research exist. Section 1.3 shortly introduces the company, Vanderlande, as part of the case study conducted for requirements and validation. The chapter ends with the approach and some basic results in Section 1.4. Here we argue how and why the chosen approach fits the research problem and explain how the research was conducted.

1.1. **Motivation and Context**

The increase of automation around the world together with the use of information systems has allowed corporations to record large amounts of data about their processes and performance. Analyzing this data can lead to valuable insights for corporations, which can support business decisions. To do so, data analytics has grown substantially and has become of more and more significance. Buzzwords like ‘big data’ and ‘data science’ are known almost to everyone these days. One of these growing data analytics domains is process mining [1]. This is specifically focused on (automatically) analyzing and improving business processes.

The general application domain for process mining as of now is based around business processes. Business process management (BPM) and business intelligence mainly focus on *information handling systems* like workflow/BPM systems, where electronic documents are pushed through an information systems according to some (predefined and documented) procedure. These procedures can be analyzed and visualized by discovering models from the real data that is logged by the systems. In this domain of process mining the cases are isolated, meaning that only one object is handled at a time. This complies to the main assumption for the majority of process mining techniques, which is that there is only a *single case identifier*.

Attempts to apply process mining on non-isolated cases, like handling business objects in workflow/BPM systems, have led to the rise of artifact-centric process mining [2]. Consider cases where many different documents or data objects are involved or where there is a complicated interplay of multiple different processes, which leads to having **no clear definition** of a single case identifier to be used in process mining, thus violating the main assumption of most process mining techniques. **Artifact-centric process mining** solves this by discovering separate **artifacts** with their own **lifecycle**, each with their own (case) **identifiers**, and then discovers the **interactions** between these artifacts.

Next to information handling systems there also exist **material handling systems**. These are systems that, as the name suggests, automatically handle materials in multiple different ways. In general there are physical items moving on conveyor belts to get from point A to B. More explicitly, a material handling system is a complex network that moves large amounts of materials from various entry points to various exit points, including routing, processing, storage, and grouping in batches. Applications range from sorting systems to fully automated warehouses. In these systems cases are not only linked on the process level but also physically. The physical flow of items along the system can interfere with the flow of other objects and has to be accounted for. Moreover, most of these systems have not been designed with data management and *logging* in mind and are mostly optimized for performance. Thus, applying process mining on these systems is not trivial.

The described application domains of process mining can be summarized into four quadrants, as visualized in Figure 1. A differentiation is made on two levels: single vs. multiple case identifiers, and information vs. material handling systems. The first quadrant, in the upper left corner, is the ‘standard’ application domain for process mining, where a clear single case identifier exists for an information handling system. Think of systems like business process management systems which can enforce workflows for cases. In this quadrant there are many solutions for process mining: multiple process discovery techniques [1] [3] [4] [5], industrial tooling such as Disco [6], research tooling such as ProM [7] [8], and modeling languages like BPMN [9] and Petri nets [10]. Many more solutions exist, too
much to name, and still a lot of research is being conducted on this quadrant as it is the most adopted use of process mining in the industry.

The second quadrant, in the bottom left corner, covers information handling systems where multiple case identifiers exist. There are a few solutions that can handle this, like the previously mentioned artifact-centric process mining approaches [2]. More specifically, for ERP systems, semi-automated ways of extracting and analyzing event logs (and models) exist [11]. More general ways of extracting artifact-centric logs are also available, but the assumptions for the input of such approaches are still quite considerable [11] [12]. The existing solutions are far from being adopted in practice as well as still having many implications. Research is still being conducted in this area for these reasons, with many more solutions on the horizon [2].

Figure 1: Four quadrants of systems and case identifiers

The third quadrant, in the upper right corner, covers process mining for single case identifiers in material handling systems. Baggage handling systems at airports and package sorting systems are examples of material handling systems. There is very little research done in this domain and as such it is not clear if process mining works well in this area. However through other projects at Vanderlande a new technique called the “performance spectrum” has been developed [13]. It has been shown that process mining can work well on these types of systems, given the right set of tools and pre-processing steps. Tools like the Performance Spectrum Miner [13] and Disco [6] can be used to gain valuable insights.

The final quadrant, in the bottom right corner, covers process mining for material handling systems with multiple case identifiers. Examples of this are warehouse automation systems and baggage handling systems. However, even though baggage handling systems do include batching, it is less impacting and prominent, and thus we choose to focus on warehouse automation systems in this thesis. Warehouse automation systems are a subset of material handling systems that automate processes within warehouses, which also include automated batching. The system studied in this thesis consists of multiple sub-systems each handling its own functionality and responsibility as part of a whole. The software running on top of this system is similarly split up into multiple modules and one super module. Objects are moved through the system on conveyors and brought to the correct module or place in the system for further handling. In short, a warehouse automation systems fulfills shop orders by consolidating multiple different products onto one pallet as follows:

1. Pallets with products are received and temporarily stored
2. Pallets are de-stacked and the products are put in individual trays (which generally hold 1~4 products)
3. The filled trays are temporarily stored
4. The filled trays are retrieved from storage and the products are used to stack an order
5. The finished stacked pallet is brought to the loading dock of the warehouse, where it is loaded into a truck and sent to the shop

Hence, products are handled on pallets and on trays and the handover is done automatically. Each tray and pallet has its own identifier and processes need to be analyzed across different pallets and trays, following the process. This yields multiple case identifiers and complicated interactions between processes. Moreover, the data structures and models behind warehouse automation are not focused on or built for process mining, which we will elaborate in Chapters 3 and 4. General use of orchestrations and unclear lifecycles of objects lead to challenges for current process mining applications. We give a more elaborate explanation and example of warehouse automation in Chapter 3 to analyze clearly where warehouse automation systems do not fit process mining assumptions.
As has been stated, no process mining techniques have been applied on such systems yet. We therefore decided to conduct an exploratory research project to research the influence of physical flows and interactions between multiple case identifiers on a relevant class of warehouse automation systems.

To apply existing process mining techniques on warehouse automation systems the first step is getting an event log, as data is not logged in the shape of an event log, which we will examine more in Chapter 5. However, due to the nature of these systems as previously described, extracting an event log is not a trivial task. Traditional log extraction approaches fail, solely due to the fact that the assumption of a single case identifier is violated. Introducing multiple case identifiers can lead to issues such as data convergence and divergence. These can potentially be solved by using artifact-centric process mining approaches but this might not always be applicable. Hence, a need for an extraction method arises that can account for multiple objects, their relations, and their lifecycles.

On top of log extraction, model analysis and pre-processing also poses an interesting challenge. Due to the relation between pallets and trays that is a result of the batching, the case identifier is not clearly defined and thus we cannot simply use this as-is in existing tools. Moreover, due to the intricacy of the system there are many long, highly variable traces, with many different events and possibly multiple objects in one trace. Interpreting 1:n relations between processes or activities in process models is also no trivial task. Divergence introduces false edges in the model and thereby skews statistics, whereas convergence leads to more events and flow than is actually in the data. For instance, applying a process discovery tool for a single case identifier on a log with divergence and convergence results in a model that has 49% false edges in comparison to an artifact model that has none [11]. The proposed solution solves both divergence and convergence but is not literally applicable in our case, as we will further elaborate in Chapters 3 and 4.

1.2. Research Problem and Methodology

As discussed in Section 1.1, process mining on material handling systems with batching faces challenges and problems from the 3 other applications domains we discussed in Figure 1 combined. It may therefore be possible that solutions from the other three application domains can be applied on the fourth domain. Therefore, in this explorative study we aim to figure out if and how we can apply the lessons learned in the other quadrants so that we, and specifically the engineers at Vanderlande, can successfully apply existing process mining techniques on data of warehouse automation systems. Thus, we define the research problem as follows:

*Given data of a material handling system with automated batching and multiple different case identifiers, we would like to devise a methodology so that engineers (at Vanderlande) can apply current process mining techniques to do analysis.*

It is assumed that the data at least contains a timestamp, an identifier, and something that resembles an activity. Moreover, it is assumed that the material handling system is documented well and that the data is readily available in some sort of database.

As the research problem is defined on a relatively high-level we split it into parts, which each correspond to a separate research endeavor. As such the resulting research questions (RQ) are as follows:

(I) *Given a material handling system with automated batching, including multiple different case identifiers, we would like to gain conceptual understanding of the processes and the difficulties it brings for log extraction.*

(II) *Given the conceptual understanding of the processes within a MHS we would like to devise a way to extract event logs.*

(III) *Given a set of event logs that contain data of a MHS we would like to explore the possibility of applying and adopting current process mining solutions to attain insights in the applicability and limitations of these solutions.*

(IV) *Given the insights in applicability, and the limitations of the current process mining landscape, we would like to distill a methodology for applying process mining on MHSs and we would like to identify future research problems.*
The objective is hence not to get new discovery algorithms or visualizations in tooling, but to get understanding of the limitations and challenges this application domain creates for process mining. The outcome should create a clear path forward for future research together with a methodology to obtain event logs and do analysis with existing tools, highlighting what functions well and where limitations are met.

Next, we discuss the methodology we followed to tackle the research problems. We split the research into a few phases, broadly following CRISP-DM, as proposed in [14]: exploration (RQI), log extraction (RQII), log pre-processing and analysis (RQIII), evaluation (RQIII and IV), and finally conceptualization (RQIV). Relating this to CRISP-DM, exploration covers business and data understanding, log extraction (and pre-processing) is data preparation, log (pre-processing and) analysis is modeling and evaluation. Please note that after each result is obtained a minor evaluation was done, meaning that evaluation practically comes into play after each other step. We do not deploy anything but this is replaced with the conceptualization step. An overview of the resulting approach is shown in Figure 2, where each step is labelled with the corresponding research question it aims to answer.

Figure 2: Cross-industry standard process for data mining (CRISP-DM) model and the slightly adjusted version

This way of working allowed for quick iterations and necessary feedback loops. Especially at log pre-processing and analysis a lot of reiteration had to take place and also between data understanding and data preparation, as working with the data led to a better understanding. As such, each phase answers a part of the research question, with the first phase being an essential prerequisite of the research endeavors. How we tackled each phase and which results came out of it is discussed in Section 1.4, after we introduce the case study in Section 1.3.

1.3. Case Study

The research in this thesis is conducted on a warehouse automation system designed by Vanderlande Industries B.V. (hereafter: Vanderlande). The results of this research have also been validated through a case study on that specific warehouse automation system of Vanderlande. Vanderlande is a global market leader that supplies all types of automation for internal logistic processes, from baggage handling at airports to warehouse automation.

Vanderlande focusses on optimization and automation of existing logistic processes by closely working together with its customers before, during, and after the execution of a project. The company has achieved a revenue of over 1 billion euros and employs more than 5000 highly skilled workers worldwide [15]. The headquarters is located in Veghel, where it was originally founded, where over 1500 employees are based. Vanderlande operates in four main market segments: airports, warehousing, (postal and) parcel, and life-cycle services. In each of these segments there are many different automation solutions which are usually co-created with the customers to achieve the highest level of satisfaction and efficiency. The focus of this research is in the warehousing market segment. This segment is one of the newer segments in terms of completely automated systems, as it takes over the entire responsibility of a warehouse, including planning.

The system studied in this thesis is a warehouse automation system that handles products on multiple objects: pallets, trays, and roll-cages or pallets again. Throughout the system the products are taken off or put on one of these objects and will always be transported while in or on such an object. The system operates based on orders which is regulated by a software package. The software system is capable of executing a complete in to out flow in
the warehouse. Hence, the system takes over the entire responsibility of a regular distribution center. As such, this system is a perfect concrete embodiment of the mentioned material handling systems, which will become more clear in the following chapters.

In Chapter 3 the processes and concepts of warehouses in general and a warehouse automation system are explained using an example. After this is clear, in Chapter 4, models are drafted that help with understanding the challenges for log extraction approaches. These examples will provide a basis for the entire thesis as it will help explain why certain decisions were made. Note that the example is not an exact representation of reality but is an abstract limited to only the concepts that are necessary to understand the reasoning in the later chapters.

1.4. Approach and Results

In this section we describe the approach we used, more specifically how we answered the research questions. Each step of the methodology shown in Figure 2 has its own outcome as is shown in Table 1. We go over each of the research questions stated in Section 1.2 one by one and explain how we reached an answer, shortly discussing the result as well.

Table 1: Outcome for each of the taken steps in the methodology

<table>
<thead>
<tr>
<th>Methodology Step</th>
<th>Outcome</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Understanding</td>
<td>Conceptual models of warehouse automation</td>
<td>3 and 4</td>
</tr>
<tr>
<td>Data Understanding &amp; Preparation</td>
<td>Log Extraction</td>
<td>5</td>
</tr>
<tr>
<td>Model Analysis &amp; Evaluation</td>
<td>Selection of techniques &amp; tools from results of case study</td>
<td>6 and 7</td>
</tr>
<tr>
<td>Conceptualization</td>
<td>Methodology + follow up research questions</td>
<td>8 and 9</td>
</tr>
</tbody>
</table>

(I) **Given a material handling system with automated batching, including multiple different case identifiers, we would like to gain conceptual understanding of the processes and the difficulties it brings for log extraction.**

To answer this first defined research question, in order to identify where existing techniques apply and where limitations might be met, it is essential to understand what we are looking at. Thus, after first exploring the preliminaries and related work, which we describe in Chapter 2, we did explorative research on the Vanderlande system. Research was carried out by reading documentation, exploring the available data, and interviews or meetings with experts. We validated the resulting models with domain experts and modeling experts to ensure they were adequate. We drafted conceptual models of the system and the processes pertaining to it as a baseline, which we discuss in Chapters 3 and 4. Note that the models in Chapters 3 and 4 are not the actual results, but are examples that show the required concepts of warehouse automation. Together with the definition of use cases and lifecycles, the models answer our first research question, as they grant us conceptual understanding of the processes and the difficulties it brings for log extraction.

(II) **Given the conceptual understanding of the processes within a MHS we would like to devise a way to extract event logs.**

Following up on the conceptual understanding of the system and corresponding processes, to answer the second research question, we devised a way to extract logs – as is described in Chapter 5. The traditional approaches for log extraction could not be applied, and thus a new methodology had to be devised. Data was extracted from a Splunk [16] instance using the Search Processing Language (SPL), a query-like language inherent to Splunk. First, we took a simple approach where we used the available identifiers as a case identifier, which – as expected – failed, because there was no clear description of a case. To define a notion of a case, using the use cases and the lifecycle models from the previous chapters, we then explicitly specified the lifecycle of objects, and with that the case identifier of the processes. We then used this to revise the log extraction approach. The resulting extraction method consists of two parts. For the case to be extracted a log extraction specification (LES) that links objects together within a timeframe was defined, consisting of identifiers and timestamps, ensuring well defined and bounded lifecycles for each object. Using this LES the event log was extracted by extracting 1 trace at a time. Extracting the logs in this fashion ensured proper case identifiers for each trace and event. Moreover, we defined various case id types in the first step, i.e. including 1:n relations or not. We used these multiple case identifiers by selecting a different case id type as the case id to be used during model discovery. We will explain the details of these case identifiers in Chapter 5 as well.
(III) Given a set of event logs that contain data of a MHS we would like to explore the possibility of applying and adopting current process mining solutions to attain insights in the applicability and limitations of these solutions.

To answer the third research question we first examined some initial results using Disco [6]. The main problems were the size of the model and the number of interactions (often leading to so-called spaghetti models). Having identified the main challenges we proposed a number of approaches to improve the model’s readability and thereby its value for analysis. In general these are adding context to labels, aggregation, filtering (projection and selection), and splitting up the logs, which we describe in Section 6.3. Then, we identified existing tooling that can also help us solve some of these challenges, two of which were inevitable as they are used within Vanderlande: Disco and the Performance Spectrum Miner [13]. The other tools are selected based on their capabilities: Inductive Visual Miner [17] [4], Interactive Data-aware Heuristics Miner [5], and Log 2 Model Explorer [18], which we describe in Section 7.1. Together with experts and engineers at Vanderlande we then identified criteria for evaluating these improvement approaches. To limit the number of combinations we did a combinatorial study, where we combined a number of different event logs with certain approaches and/or tools to get a selective list of results, the specifics of which are described in Section 7.2. The results were evaluated with engineers and experts at Vanderlande and with the (conceptual) understanding we have of the system and the processes. Moreover, we evaluated if the resulting models matched our expectations. We did the analysis and elaborate on the results in Sections 7.3 and 7.4. The result is a selection of tools and (pre-processing) approaches that result in better models, as defined by the criteria in Section 6.1. Through the evaluation of the models we discovered that not all tools work as expected and that some pre-processing approaches work better in certain situations. More specifically, we identified when and how the improvement approaches can be used by engineers to answer analysis questions. Next to that, we identified where limitations are met and future work is necessary.

(IV) Given the insights in applicability, and the limitations of the current process mining landscape, we would like to distill a methodology for applying process mining on MHSs and we would like to identify future research problems.

Finally, to answer the last research question we had to do two things. First, to devise a methodology and guideline we conceptualized the steps we took to reach results one by one. Using the gained domain knowledge, we identified which attributes were required for log extraction. Then we conceptualized the log extraction approach by abstracting from the queries we wrote to get our event logs for the case study. Finally, by doing analysis as described above, we were able to identify which current process mining tools and newly proposed improvements resulted in positive outcomes. We then documented the steps taken to reach a certain result, so that an engineer can follow these steps and obtain the same results. We evaluated the methodology by using it ourselves in the case study. The resulting methodology is elaborated in Chapter 8. In short, it boils down to gaining insight in the system and leveraging this to find objects and lifecycles in the system. Then, by drawing conceptual models of these lifecycles one can identify the required data to extract event logs. Using a two-step log extraction method as described in this research event logs can be extracted, which can then be used to discover and analyze process models using the selected tools (and the selected pre-processing steps). Secondly, we identified future research problems by looking at the shortcomings of the current solutions and those of our newly proposed solutions. We address these in the last chapter of this thesis together with some ideas and concepts that could be potential solutions to these problems. The resulting discussion of limitations and future work in Chapter 9 brings to light a clear path forward for future research and describes a few open research problems to be tackled. Hence, our four research objectives have been met and the CRISP-DM-like methodology we took to reach this has thus been effective.
2. PRELIMINARIES

In this chapter, we explain necessary concepts to address the research questions of Chapter 1 and discuss related work from which we derive possible solutions and approaches to solve the challenges that come with process mining for material handling systems with automated batching.

The chapter roughly follows the first three of the four quadrants as introduced and discussed in Figure 1. First, we discuss work for process mining in general and business processes specifically, i.e. the first quadrant, in Section 2.1. Then we discuss batching, lifecycles, and multiple objects and case identifiers in process mining. Together with that, we discuss related work for batching and artifacts in Section 2.2 – hence covering the second quadrant. In Section 2.3 we discuss previous work on material handling systems, i.e. the third quadrant, where we further introduce the available tooling as well.

2.1. Process Mining and Business Processes

In this section we first introduce process mining in general, where we also cover event logs and how they are represented in the research. Then we introduce the two tools that are used to apply process mining on the data of the warehouse automation system and how they complement each other. Finally, we discuss Extraction-Transformation-Loading, which constitutes the main concept(s) behind event log extraction.

Process Mining and Event Logs

Process mining is a set of techniques that allow for extracting knowledge out of event logs. They provide means to “discover, monitor, and improve processes” [1]. Initially, process mining assumes event logs already exist, i.e. software systems record data in such a way that the data can be used as an event log as is. However, as this is not the case for the warehouse automation system (event) log extraction has been added to the scope of process mining. This is shown in Figure 3. Log extraction is the process of data preparation and extraction such that an event log is obtained.

An event log consists of lists of events. These lists are called traces, which consist of a list of tasks (events) that are executed for a specific run through the process. Each of these tasks or events contains at least the following attributes: case identifier, timestamp, and activity. A case identifier uniquely identifies a specific instance of the process (a trace) such that the correct events can be grouped together. A timestamp holds the time at which a specific process step occurred, potentially split up in start and finish. An activity is a description of the process step or task that was executed. An event can hold many more attributes, which can be used as additional information, like who executed the task (a resource). An example of an event log is presented in Figure 4 and Figure 5. Events can be ordered and presented in multiple ways as we show. Note that the event log can also simply hold a list of events, for instance in a CSV format, as long as each of them has a case identifier. The case identifier can then be used to build traces, as is shown in Figure 5. In this thesis two formats are used: CSV and XES [19]. In the XES format the event log consists of traces, in which the events are already grouped together and sorted.

Figure 3: Process Mining (red) and log extraction (green), as described in the Process Mining Manifesto [1]
Furthermore, the process mining scope in this thesis is mostly limited to log extraction and (process) discovery. This is due to the explorative nature of this thesis and the current state of process mining within Vanderlande. To discover a process model the event log has to be processed by a so-called discovery algorithm. A process discovery algorithm takes an input and event log and outputs a process model. There are many different algorithms, some common ones being the Alpha algorithm [3], Inductive (Visual) Miner [4] [17], (Flexible) Heuristics Miner [20], and ILP Miner [21]. Each of these algorithms can output a different model than any other algorithm. This is because each algorithm has its own definition to determine the causality between two activities (i.e. when two activities follow each other). Moreover, not all algorithms output the same type of process model. Some output a Petri net or Workflow Net as described by W. van der Aalst in [10], whereas others output a simple directly follows graph. Most algorithms tend to output a ‘main’ flow, i.e. the most occurring flow in the data, optimizing and balancing a few process model quality criteria like precision and fitness as described in [1]. To allow for more specific analysis most algorithms also have different input variables that allow for discovery of different models. This has to be taken into account when using any discovery algorithm. In this thesis the main two tools used for process discovery are Disco [6] and ProM [8] [7], as described in the following section.

Disco and ProM
To evaluate the process discovery algorithms we propose in Chapter 6 and the case study results in Chapter 7, we make use of two process mining tools that are currently used within Vanderlande, which we briefly discuss next. Disco is a process mining tool with a freemium model, which allows for quick and easy process discovery and analysis. Its features cover but are not limited to: automated process discovery, process map animation, detailed statistics, importing and exporting data or the process maps in multiple formats, project management, and filtering of the data. Hence, Disco offers quite a complete package for quick and easy process discovery and analysis. Disco discovers directly-follows graphs, which are directed graphs that connect two activities (nodes) with an arrow (edge) whenever they happen after each other. The user can select the level of causality between two nodes, and the number of activities (nodes) to include in the process map using two sliders from 0% to 100%. An overview of the Disco GUI can be seen in Figure 6.

---

**Figure 4**: Event log example as a table in Excel, sorted on time (left), and sorted on CaseID (right)

**Figure 5**: Event log example in CSV format (left), and other representations of event logs (right)

Event logs are also often represented as a set of traces: $L = \{(A, B, C, D), (A, C, D), (A, A, B, D)\}$.
ProM is a framework that includes many plugins that allow for a large variety of functionality related to process mining, which can be downloaded free of charge on their website [7]. As such, multiple discovery algorithms can be used. This is the main use case for ProM in this thesis. More specifically, ProM will be used for the following discovery algorithms: Inductive Visual Miner [4], Interactive Data-aware Heuristics Miner [5], and Log 2 Model Explorer [18]. We will elaborate on these algorithms in Chapter 6. Finally, both ProM and Disco can be used to transform CSV event logs to XES event logs, which is a necessary step to further process the event logs. In this study we aim to investigate how these process mining tools can be applied on warehouse automation systems.

**Extraction-Transformation-Loading**

Extraction-Transformation-Loading (ETL) comes into play when an event log is not available yet, but first has to be extracted from some system; a problem we also face in this project. ETL is a concept describing three steps usually taken to combine data from multiple sources [22]. As the name suggests, data is first extracted from one or more sources. The data is subsequently transformed, i.e. preprocessed into a desired format for analysis, and finally loaded (stored) into any database or filesystem. We need to follow an ETL approach to turn the source data into something that can be stored as an event log, after which it can be loaded for analysis.

An exploratory study by S. Sharma and R. Jain describes an overview of tools and approaches to apply for ETL [23]. The three main approaches of setting up an ETL environment are mapping expressions, conceptual modeling, and UML modeling. They compare the three and conclude that using a conceptual approach for ETL modeling is best. Trivially, this means it is best to first have conceptual diagrams or models of the data sources one wants to combine, then set up a conceptual ETL framework in between, and finally build it. Another study by R. Wijaya and B. Pudjoatmodjo walks through this process and implements a ETL framework [24]. Finally, P. Vassiliadis et al. provide an in depth explanation of this conceptual approach for modeling ETL activities. Moreover, they provide a formal foundation for the concepts. The final conceptual model can be customized and extended where necessary.

ETL is an important step when consolidating data from multiple sources. However, we assume that the data from the systems we are studying resides on either a single source or resides on multiple sources that are accessible via one access point. Hence reducing the need to manually combine data from multiple sources. In the specific case of the warehouse automation system studied in this thesis the data from all the different (physical) modules is logged to the same instance of a data warehouse. The data warehouse automatically performs ETL based on how it was set up. Thus, we do not need to apply ETL to gather data, but we still need to apply ETL. As the data is not in the form for process mining (event logs) we still need to extract the data and transform it in such a way that it is ready to be used for process mining. Hence, we need to design an ETL scheme or system to obtain event logs, which we discuss in Chapter 5.
### 2.2. Multiple Case Identifiers and Batching

In this section we discuss the implications of having multiple case identifiers and batching in the data. We start with defining batching and what it entails in the setting of warehouse automation systems, after which we discuss some related work on batching in process mining. Then we specify what we mean when we refer to a lifecycle throughout the thesis. Finally, we discuss artifact-centric process mining and how it can help us solve the challenges as described in Chapter 1.

**Batching**

Batching or to batch is described as “to make a group out of a number of things so they can all be dealt with together”, and a batch as “(…) things dealt with as a group or at the same time”, in the Cambridge Dictionary [25]. In the context of this thesis we look at ‘systems with automated batching’, hence a system that at some point in its lifecycle groups together a number of things to handle them at the same time. Typically, batching in process automation is used to execute a number of (potentially different tasks) at once using only one resource. For instance, an acceptance email is sent out to multiple applicants after checking their information. However, in this thesis batching is not so much about executing tasks but is based on physical batches.

The batching in a warehouse automation system is based on products. Products are on a pallet at first, where they are all handled at the same time, later the products are put in trays, then finally the products are put back on a pallet where they are again handled in a group at the same time. Hence, the system automatically includes batching and ‘de-batching’ i.e. to split up a group of things that were previously handled together.

Batching brings with it an interesting challenge as shortly mentioned in Chapter 1. Process mining assumes the existence of a properly defined case identifier. However, with batching one has a multitude of choices for the case identifier: the entire group, a subset of the group, or one item. Moreover, as in the warehouse automation system items are first de-batched and later batched again there are two batching moments which both impact the selection for the case identifier. Moreover, this leads to 1:n relations between some objects (with their own identifiers) and can hence lead to activity duplication in the event log if not handled with care [26].

In literature the concept of batching is handled as well, albeit a bit differently. Luise Pufahl and Mathias Weske introduce the concept of batch activities in their paper [27]. They cover process modeling and execution of these batch activities. The aim is to synchronize execution of a group of instances of the same (sub)process type. This is achieved through assigning a batch model to an activity, which turns it into a batch activity. The batch model describes several parameters that have to be configured properly by the designer of the process and as such is not trivial and is not automated. A lot of care has to be taken when designing process models with these batch activities.

The same authors, joined by Andreas Meyer, extend their previous work on batch activities by introducing a larger concept called batch regions [28]. This is done by extending one batch activity into a larger set of connected activities, thus becoming a region in the process model. On top of this they also introduce data views which take from the database domain to help specify how data is used to determine which activities or regions can be synchronized (or batched). Furthermore, this allows for further detailing which specific instance of a same process can or cannot be synchronized, based on the data attributes (i.e. a differing shipping date). Again the concepts are studied for modeling and execution, however no specific implementation is given, which makes trying out the concepts relatively cumbersome.

The concepts of batch activities and regions seem promising for our research. However, the focus of these is to synchronize process execution across multiple instances of the same process. In the warehouse however this is not possible and necessary. Synchronization automatically ‘takes place’ by having the products on a pallet and hence reducing the number of process instances necessary to transport all these products (i.e. 1 pallet vs 100 products). Moreover, it is not our intention to synchronize process execution or to reduce process execution costs. The aim is to analyze the batching that is already present properly and to be able to generate insights. Lastly, using batching regions or activities is not possible as it is necessary to handle explicit trays or pallets separately and one cannot just arbitrarily combine the products in them based on data. Moreover, this ‘synchronization’ is already done beforehand, where it is determined which pallets and trays are necessary to fulfill an order, and as such there is little optimization to achieve when using batch activities or regions. The challenge in our case rather lies in defining a notion of a case, and thereby defining a lifecycle for each object.
Lifecycles
Throughout this thesis we will often refer to the notion of ‘lifecycle’. Defined as “the series of changes that a product, process, activity, etc. goes through during its existence” in the Cambridge Dictionary [25], we refer to this lifecycle as the process steps an object or item goes through during a process instance. Moreover, an item may not have a fixed, single lifecycle, but it may follow different lifecycles based on the process it participates in. This means that the process (instance) defines the start and end of such a lifecycle, if there is any. Tying back into the dictionary definition, the existence of a process mining case is bounded by the process definition and the corresponding process instances of said process. In simpler wording: an object may have multiple different lifecycles, each corresponding to a sub-process of the complete process, each of which has a start and end defined by the activities in the respective sub-process. As mentioned in the introduction, we need to specifically determine the lifecycles for the warehouse automation processes and the objects that are handled within such a system. We do so in Chapter 3, which explains the system and the corresponding processes, and in Chapter 4, which provides process models that are specifically designed to show the lifecycle of each object for a given process instance.

Artifact-centric Process Models and Mining
Artifacts are described as “a mechanism to record (...) information in units that are concrete, identifiable, self-describing, and indivisible” by A. Nigam and N.S. Caswell [29], or more recently as “entities described by both an information model and a lifecycle” by X. Lu in her journal paper on artifact-centric process mining [11]. Artifact-centric process models can solve convergence and divergence issues in information handling. In the paper by D. Cohn and R. Hull [30] an approach to modeling processes using business artifacts is introduced. An overview of the current state of discovery algorithms for ‘artifact-centric process mining’ can be found in the similarly named paper by D. Fahland [2], which is explored more later. These so called artifact-centric process models describe a process as a collaboration between multiple artifacts with their own lifecycle. To discover an artifact-centric process model, first all separate artifact schemas have to be discovered from the data, after which the lifecycle of each artifact can be discovered. Then, the interactions between the discovered artifacts can be discovered using various existing discovery algorithms. Several other researches have used artifact-centric approaches in their endeavors to use process mining on amongst others ERP systems, as is elaborated below.

The first step towards solving the issue of having multiple case identifiers was proposed by A. Roest in his master thesis, where he proposed to combine several identifiers into one case identifier [31]. In this case it was for the Order-To-Cash process of SAP, where he used all primary keys of the corresponding tables. However, this approach is obviously very specific and does not translate well to automated warehousing. Moreover, the notion of a case depends on what the user wants to investigate and hence can vary. Nonetheless, combining several identifiers into one, including enriching the traces, can pose to be a good starting point for the log extraction of data with multiple objects.

Combining multiple case identifiers can also be done by enriching the trace towards the lowest level of case identifier. In our case this would be towards the tray, which has the highest multiplicity/finest granularity. This was proposed by J. Buijs in his master thesis as a partial solution to convergence and divergence [32]. However, doing so will not completely solve convergence and divergence. Specifically when enriching traces by adding higher level events to lower level events, these added events are duplicated for every lower level event they have a relation to. Furthermore, without enriching traces some events are simply left out, therefore not necessarily solving convergence but more so avoiding it. Hence, extra care has to be taken with choosing a case identifier and with trace enrichment.

E. Nooijen et al. were the first to introduce an (semi-)automatic way for artifact-centric log extraction. It was introduced in his master thesis, which was later followed by a published paper [33] [34]. This is done by first discovering the artifact schemas and afterwards discovering the lifecycle of each schema separately. The resulting framework and implementation was further developed by X. Lu in her master thesis and a follow up paper [26] [11]. An artifact-centric process model describing the business objects, the corresponding lifecycles, and the interaction between these objects during their lifecycles can be obtained semi-automatically as a result of the endeavors. The framework is specifically focused on extracting artifact schemas and lifecycles from relational databases, like in ERP systems. This makes it quite difficult to translate this approach directly onto the warehouse automation systems (and data), as this is not stored in a regular relational database. However, the changes to objects do reside in multiple different “tables” and one could consider the different processes at each module an artifact. On the other hand the artifact schema identification would have to be done manually as well as the lifecycle ‘discovery’, as the developed tooling is not available and we do not have such specific relational data. Hence, the concepts and underlying
approaches can be used, but the framework would require significant changes and (re)development to work with different data.

Another paper on artifact-centric lifecycle discovery is that by V. Popova et al. [12]. It presents a chain of methods (and tools) for discovering artifact lifecycles. While doing so, the problem is decomposed specifically so that multiple different existing process discovery methods and tools can be used. Hence, it allows for picking a discovery algorithm that is most suited for the data and process. First, artifact-centric logs are build, after which a few different miners are used to discover the artifact lifecycles. Finally, the discovered Petri nets are translated to Guard-Stage-Milestone models. The discovered models only focus on one artifact, and it is not yet possible to discover interactions between artifacts like with Lu’s work. Thus, as we are interested specifically in the interactions between the artifacts this work does not yet provide an as-is solution to the research problem.

To round up the discussion on artifact-centric work we revisit the previously mentioned pre-print paper by D. Fahland, which consolidates the current state-of-the art in discovery for artifact-centric process mining [2]. The main takeaway relating to this thesis is the discovery of artifact schemas and their lifecycles. This can be done if the input is in the right format: either a log with events carrying information about the data objects and their changes, or a relational database (like with ERP systems). Achieving such a log is not trivial for the data model of a warehouse automation system as described. Moreover, if the objects in the system – tray or pallet – become the artifacts in consideration the resulting models will be very large as the lifecycle of these objects is long. Furthermore, the lifecycle of a tray has to be specified explicitly as in the data it has none. If the modules in the system become the artifacts there will be very little interaction between them. It is however good to model these modules as separate entities from each other (like they are in reality). These models can then be projected onto either a tray or pallet to find the ‘artifact-centric’ lifecycle model of either. However, artifact-centric models can be ‘discovered’ manually, as well as the lifecycles, based on domain knowledge. We describe models that are inspired by the artifact-centric approach in Chapter 4. These will pose a direction as to what kind of challenges we face for log extraction and finally model discovery and analysis.

### 2.3. Material Handling Systems

As we mentioned in the introduction, previous projects at Vanderlande have resulted in the development of a new process mining technique called a ‘performance spectrum’. An example of such a spectrum can be seen in Figure 7, which is discovered using the Performance Spectrum Miner [13]. The Performance Spectrum Miner was originally developed for baggage handling systems to identify performance issues by tracing each bag separately. The performance spectrum is read as follows: each horizontal ‘bar’ represents a segment in the data, i.e. an edge from activity A to activity B. The colored vertical bars in each segment show the number of items/objects that were on this segment in a certain time frame (the black-white bar on the bottom), relative to the maximum, which is listed next to the segment name on the left. The colored (slanted) lines show individual traces.

![Figure 7: A performance spectrum, as discovered in the Performance Spectrum Miner](image)

Together with the Performance Spectrum Miner (PSM) a pre-processing framework was developed to structure the data in such a way that it can be handled by the PSM. This framework is also used for pre-processing of our data, not only for the PSM but also in general, as we discuss in Chapter 6. Through the PSM project it was also made possible to analyze event logs of this type of material handling systems in Disco. However, the PSM has not been tested with data that includes multiple case identifiers and a high degree of (automated) batching, as is the case for data of warehouse automation systems. Hence, we cannot simply assume that it can be used for analysis purposes, and thus we have to explore the possibilities.
Next, we will explore how or whether all the techniques mentioned in this chapter carry over to the fourth quadrant. We explore artifact-centric ideas for modeling in Chapters 3 and 4 and for log extraction in Chapter 5, and explore classical process mining (i.e. Disco and ProM) and MHS mining (PSM) for discovery and analysis in Chapter 6 and 7.
3. CONCEPTUAL MODEL OF WAREHOUSE AUTOMATION

Before we can explore which process mining techniques of the three quadrants can be applied to material handling systems with multiple case identifiers (the 4th quadrant), we need to gain an understanding of warehouse automation systems (WAS), thereby answering RQ1. We achieve understanding of warehouse automation systems by conceptualizing it and explaining the processes within the system. Hence, for the sake of clarity and to explore and explain all the concepts that underlie or entail the challenges in this thesis we create a toy example for a warehouse and a warehouse automation system. This example will be elaborated in a few steps. First, in Section 3.1 the general processes in a warehouse are discussed, together with the challenges this entails. Then in Section 3.2, a warehouse automation system is discussed that executes these processes. The example will be referred back to throughout the entire thesis as it forms the precedent for the solutions and approaches discovered further on.

3.1. Warehouse Processes

Before the exact steps are discussed let’s first take a look at the goal or mission of the warehouse. The warehouse supplies shops with requested goods from various different vendors, which is supplied in a shop order. The order is fulfilled by consolidating a set of goods from (often third party) vendors from separate pallets to a single pallet (or set of pallets) specifically for the shop. These pallets are then shipped to the respective shop. More often than not a multitude of shops are in the same shipment. The goods from vendors can be stored temporarily in the warehouse for up to around a week, whereas the finished shop orders normally only remain in the warehouse for up to a few hours.

Now that it is clear what the warehouse aims to do let’s look at how it does so. Figure 8 lays out the high level overview of the process. The process can be cut up into four main sub-processes: inbound, de-stacking, stacking, and outbound, which are identified in Table 3.1. In a warehouse multiple instances of these four sub-processes run in parallel. The end-to-end process for any given product is the succession of the four sub-processes, with the transport process interleaving, i.e. as described in Figure 8. The following will describe this process step by step, every time referring to the numbering as in the figure.

![Figure 8: An overview of the process in an automated warehouse](image)

### Table 3.1: Steps and their respective encompassing (sub-)process and involved modules

<table>
<thead>
<tr>
<th>Process</th>
<th>Steps</th>
<th>Involved modules*</th>
<th>Involved Object</th>
<th>Covered/Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inbound</td>
<td>1, 2, 3</td>
<td>Pallet storage</td>
<td>Pallet</td>
<td>✓ (except 1)</td>
</tr>
<tr>
<td>De-stacking</td>
<td>4, 5ab, 6</td>
<td>De-stacker, Tray storage</td>
<td>Pallet, Tray</td>
<td>✓</td>
</tr>
<tr>
<td>Stacking</td>
<td>6, 7</td>
<td>Stacker, Tray storage</td>
<td>Tray, Pallet</td>
<td>✓</td>
</tr>
<tr>
<td>Outbound</td>
<td>8, 9, 10</td>
<td>-</td>
<td>Pallet</td>
<td>X</td>
</tr>
</tbody>
</table>

*The transport module (and process) is involved in all tasks as it is the glue between all modules.

The process ‘starts’ when a truck arrives at the warehouse (1). The truck is then unloaded either manually or automatically using conveyors (2). The pallets are then transported from the dock towards the pallet storage (2), where they are stored (3). Here the pallets reside until they are needed for a shop order and are then retrieved from...
the storage (3) and brought to the de-stacker (4). In case of a manual warehouse this means the pallets are placed on ground floor where workers can take the products from them. However, in an automated warehouse the products usually have to be taken off the pallet and are generally put into smaller containers (trays) (5b). Concurrently, the empty pallet leaves the system (5a). The products, which now reside in the trays, are transported to storage (5b), where they are subsequently stored (6). Here the trays reside until all products for the shop order are in stock (i.e. in the tray storage); this is called order consolidation, which is often done to improve throughput times – even in manual warehouses. When the order is consolidated the trays are retrieved from the storage (6) and brought to the stacker (7). The stacker automatically stacks the products onto the pallet (7). The completed pallet is then transported to the loading dock (8). Here it resides until it is loaded into the truck (9). Once the truck is full, i.e. when the shipment is complete, it will leave towards the shops (10). In a real life situation in the system, multiple instances of this process are started perpetually as new trucks arrive and new shop orders are created.

Figure 9 illustrates the process as a partial order for an example of two shop orders. Two shops have ordered the following. Shop 1: 4 coke, Shop 2: 2 coke 6 cookies. Assume there are two pallets: P1 with 6 coke and P2 with 6 cookies. Any tray (Ta and Tb) can only hold 1 type of product and only a fixed amount, in this case 4 coke (round icon) or 6 cookies (square icon), but the tray can be used multiple times. As such, at de-stacking (step 4) the pallets will be de-stacked onto 2 trays: Ta (4 coke), Tb (2 coke), and later Tb again (6 cookies). Hence, the products on one pallet (e.g. P1) are now placed from 1 object (1 pallet) onto n objects (2 trays). Continuing the process to fulfill the shop orders at the stacker, first Tray B is used for Shop 1’s order, then Tray A, which leads to Tb being empty and Ta having 2 coke left. After that Tb is filled again at de-stacking, resulting in Tb now holding 6 cookies. Then, the order of Shop 2 is stacked: Tray A is used first and then Tray B is used, resulting in both trays being empty.

Note that if we relate this example to the steps in Figure 8 specifically and extend the example in Figure 9 to include the arrival of the pallets, something interesting happens. Consider Ta and Tb which are both filled with products from P1. The first 5 steps (1,2,3,4, 5a) of the process are only executed once, while the following steps (5b, 6, 7) are executed twice, i.e. once per tray. During step 7 the process comes together again and the final steps – 8, 9, and 10 – are executed once, for the outgoing pallet SO1. This brings us to the concept of batching, as described in Section 2.2. Consider again the relation between the pallet and a tray: from 1 pallet n trays can be filled, and n trays are used to stack 1 pallet. Thus, the process automatically handles batches and thus includes ‘batching’; the batch we refer to is a batch of multiple trays that are filled from 1 pallet or that are put on 1 pallet. Moreover, the batching is applied automatically through the change of carrier: pallet to tray to pallet (to truck). Furthermore, de-stacking of products from pallets onto trays and storing of trays has to be done for each incoming pallet, and can possibly be done in parallel if the warehouse provides capacity for this (e.g., P1 and P2 in Figure 9 could be de-stacked in parallel). On top of that, multiple shop orders can also be stacked in parallel if they do not require products from the same tray.

Hence, the (sub)processes can happen in parallel and include batching. Moreover, trays and pallets can participate in multiple shop orders (processes) and the relation of pallet:tray or tray:pallet is 1:n and n:1 respectively. We will later see that because of batching and parallelism we need to carefully define how to scope events in a case to obtain a meaningful lifecycle for each object.
Figure 9: Toy example of filling 2 trays from 2 pallets and stacking 2 orders, annotated with the process steps from Figure 8.
3.2. Warehouse Automation System

We know from the artifact-centric process mining discussion in Section 2.2 that for event log extraction in the presence of multiple objects, we need to identify case identifiers for objects, and group events into object lifecycles. Section 3.1 already identified the presence of multiple objects (pallets, trays) and complex dynamics in the process. Now, in order to define actual event log extraction, we first need to understand how these objects and dynamics are being handled and logged in the system itself. Thus in this section we elaborate on a warehouse automation system, which is the entity that moves the passive containers (tray and pallet) around. Moreover, it is essential to further investigate the different notions of case identifiers throughout the process and how they are handled, so that we can describe this behavior in terms of events and cases.

The warehouse automation system that we describe in the following section will be limited to the steps 2-7, as described in Section 3.1, and illustrated in Table 3.1. Thereby essentially leaving out the inbound and outbound process, and the ‘truck’ part, of the warehouse. This was done due to technical limitations of the system that was studied for this research. Furthermore, the scope was also partly determined by the engineers and experts at Vanderlande. They explained the most interesting dynamics to analyze, which correspond to the identified sub-processes, though including the full in-to-out process as well. Moreover, the engineers want to specifically analyze the physical flow of an object throughout the system. From these requirements we define (analysis) use cases with the same names as in Table 3.1. However, we introduce the “In-to-Out” use case, which initially covers process steps 1-10, i.e. the full process. Projecting the scope onto these use cases leads to the following definition of 4 use cases and their corresponding process steps (and participating objects):

- **Inbound**: step 2 and 3 (pallet)
- **De-Stacking**: step 4, 5ab, and 6 (pallet, tray)
- **Stacking**: step 6 and 7 (tray, pallet)
- **In-to-Out**: step 2, 3, 4, 5ab, 6, and 7 (pallet, tray, pallet)

Throughout the processes, the system handles the following different objects: a pallet, a tray, and the products itself. To identify the pallets and trays a unique identifier will be assigned to a pallet upon arrival (the Pallet ID); the trays will hold a barcode sticker that gives them a fixed unique identifier (the Tray ID). Separate products will not be tracked and hence are not given a unique identifier but do however have a code that identifies which type of product it is.

To handle the different tasks and objects within a warehouse the system is split up into several (possibly duplicate) modules, each of which has its own responsibility and task set, as described earlier in Table 3.1. The example blueprint of a warehouse automation system in Figure 10 illustrates how products arrive at the inbound door and then follow the process steps 2-7 (and out-of-scope 8 and 9) of Figure 8 until the outbound door. To connect the process step figure (Figure 8) the corresponding step numbers have been added to each of the areas. Moreover, each of the edges shows which type of object is being transported or handled in between the steps. The interplay of pallets, trays, and the remaining details on storage will be explained below.

**Figure 10**: Toy example warehouse blueprint; edges indicate what object is being moved
The system will hold a few storage ‘moments’, i.e. places in the process but also in the physical system where an object is temporarily stored. As illustrated in Figure 10 these are the Tray Storage and the Pallet Storage. However, once a tray has been emptied it does not necessarily have to go back into the Tray Storage. To optimize throughput times by reducing load on the system a special “Empty Tray Storage” is also part of the system. Only empty trays are directed towards this special storage. Lastly, there are two more unmentioned places: the empty pallet outfeed and the stacking pallet infeed. The first collects empty pallets after they have been de-stacked and the latter holds a buffer of pallets to be used to stack orders on.

The system is orchestrated by a single controller module; let’s call that the ‘conductor’. This conductor tells each module what to do when and keeps track of the state of the entire system. Communication only goes to and from the conductor, not in between modules, like in Figure 11. The conductor operates based on shop orders that come in from an outside system. A shop order is simply a list of products and amounts that a shop is requesting from the warehouse. Based on this the conductor either starts the stacking immediately or first de-stacks pallets when the required products are not in stock (in the tray storage).

The conductor gives the modules tasks to execute. These are amongst others transport, store, retrieve, de-stack, and stack. Obviously e.g. a store/retrieve task is only sent to a storage module. Each task refers to a (number of) trays or a pallet and is uniquely identified by a Task ID. Hence, through its life, a pallet or tray takes part in multiple tasks at multiple modules. For instance, an end-to-end flow for a product could look like the following sequence of tasks: transport pallet, store pallet, retrieve pallet, de-stack, transport pallet + transport tray(s), store tray(s), retrieve tray(s), stack, transport pallet. Each pallet has a unique identifier, so grouping events by the Pallet ID yields a well-defined lifecycle for a pallet participating in a process from entering to exiting the system. A tray however has a fixed persistent identifier, the Tray ID, so grouping on this Tray ID does not yield a well-defined lifecycle for the tray participating in a process. To demonstrate this, consider the following: once a tray reaches the empty tray storage it will no longer be tracked by the conductor until it exits this storage again. However, as it returns to the system, it will still have its unique Tray ID, although it is now participating in a new shop order unrelated to a previous one.

Each task is executed through a sequence of messages: Assign, Start, and End. These messages carry information about the status of the task as follows. The Assign message holds a destination for the object, the Start and End message both hold the current location of the object, and each message also holds the corresponding Task ID. During the execution of a task various location and module status update messages are sent by the modules, which we will later include in the conceptual process models in Chapter 4. Note that any messages referring to pallets do not carry information about the tray and vice versa.

Events of all messages that are sent are logged and stored in corresponding tables in the database, as illustrated in Figure 12. Each message holds at least a timestamp, an Object ID (i.e. Pallet ID or Tray ID), and a Task ID. The event type simply covers which message it is, like Assign, Start, End, or any location or module status update. Other attributes may be present in the events, but for now we do not consider these. Figure 13 illustrates two small
example tables for two event types: Transport Task Assign, and Stacker Busy (an example of a module status update message). Both event types have a timestamp, Task ID, Object ID, and some differing other attributes. In this case for the transport task the extra information tells us where the object is headed, whereas for the 'Stacker Busy' event we see how many products are taken from the object (tray). We discuss further details about event storage in Chapter 5.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>TaskID</th>
<th>ObjectID</th>
<th>Destination</th>
<th>Current Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:45</td>
<td>TT1</td>
<td>Ta</td>
<td>Storage</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:47</td>
<td>TT2</td>
<td>Tb</td>
<td>Storage</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:50</td>
<td>TT3</td>
<td>Tc</td>
<td>Storage</td>
<td>De-Stacker</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>TaskID</th>
<th>ObjectID</th>
<th>#Products</th>
<th>Current Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:03</td>
<td>ST1</td>
<td>Ta</td>
<td>2</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:04</td>
<td>ST1</td>
<td>Tb</td>
<td>2</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:20</td>
<td>ST2</td>
<td>Ta</td>
<td>2</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:22</td>
<td>ST2</td>
<td>Tc</td>
<td>4</td>
<td>Stacker</td>
</tr>
</tbody>
</table>

**Figure 13:** Example events in two different tables

Next, we investigate the exchange of the messages to explain the linkage between the different objects and tasks. As an example, a message sequence diagram [35] has been created that shows the interaction between the conductor and the modules. Messages are sent to and from the conductor in a quite sequential manner. In the case of parallel flows a “PAR” block is used, which encapsulates the two flows that run in parallel. The flow continues sequentially after such a block. The diagrams should be self-explanatory otherwise. The message sequence diagram showing only the Task messages can be viewed in Figure 14 and is annotated with the process steps from Figure 8. It illustrates the process for receiving a pallet, storing it, de-stacking it, storing 1 tray from that pallet, and using that same tray in a stacking task, which is finally transported to the dock. The Transport Task has been intentionally left out for brevity; a black box means it is synchronous, a white box means it is asynchronous (i.e. is running in parallel).

**Figure 14:** Simplified MSD for the described conceptual WAS (with process step numbers)
To further demonstrate the previously mentioned $1:n$ relation between an incoming pallet and trays, Figure 15 illustrates how multiple trays are handled in parallel after the de-stacking (again annotated with process steps from Figure 8). When all the trays are finally retrieved for the order the stacking starts. Note that in reality as soon as a tray is filled the conductor will start the Transport and Store tasks at the same time. After all the trays have been stored the order is consolidated and the retrieval process starts. Again, the trays are retrieved in parallel; although bound to a sequence. When the sequence of trays is ready the stacking starts.

We now have a description of the processes in warehouse automation systems, a description of an example physical layout, a description of how it is being operated, and we identified four use cases for analysis. We wanted to understand conceptually how warehouse automation systems work, which has been achieved through the various examples. Furthermore, we wanted to understand the difficulties these systems bring for log extraction. We established that there are multiple case identifiers due to the multiple objects, which each have their own lifecycle. Moreover, we established an understanding of the interplay of the various modules in the system and which type of data is being sent between them, and how that data (the events) is stored. Obtaining a meaningful log out of the events in the database will be challenging, as we will see in Chapter 5.

Before we can extract event logs, like we stated at the start of this section, we need a specific definition of the lifecycle of each object (per use case). Hence, what remains to be done is identifying these lifecycles, so that we can use this information to apply artifact-centric thinking to set up a log extraction approach, using an ETL approach as elaborated on in Section 2.1. We aim to specify the lifecycle of each object by creating a model based on the physical flow per object, which we discuss in Chapter 4.
Figure 15: Simplified MSD showing the 1:n relation between a pallet and trays (with process step numbers)
4. **LIFECYCLE MODELS**

In Chapter 3 we explored a warehouse automation system and identified the various objects and identifiers in the processes of the system. This puts it very close to artifact-centric process mining as described in Chapter 2. We also discussed that the system is designed in a systematic way, i.e. split up in modules, and that trays and pallets follow a certain structured behavior. We captured this behavior in some examples, especially Figure 9, Figure 14, and Figure 15, which illustrate the behavior in various ways. As we shortly stated at the end of Chapter 3, if we can describe this behavior in “lifecycle models” we can apply artifact-centric thinking to approach the problem of process mining (and specifically log extraction) for these warehouse automation systems. Thus, in this chapter we aim to manually specify the lifecycle models of both a pallet and a tray. Moreover, we want to identify whether we must describe these lifecycles in terms of individual object lifecycles, potentially split into the separate modules, or if we can combine them and describe the process as a whole. The lifecycles we are going to specify will later help us with defining a log extraction approach, specifically an event log extraction specification in Chapter 5, which will help us extract meaningful event logs. Moreover, in Chapter 6 when we explore the data we can use these lifecycle models as a baseline of what we expect to find. Finally, the lifecycle models will also help us further specify the use cases we defined in Chapter 3.

4.1. **Modules vs. Complete Process**

We explored several options for modeling the behavior of pallets and trays using BPMN and Petri nets, and evaluated models regarding the separate modules as introduced in Chapter 3. We then concluded that plain end-to-end models that cover only separate modules do not give us enough insights in where the lifecycles of especially a tray start or end with respect to the full process. The models are interesting and creating them is a good exercise for gaining better understanding of their behavior, but they do not show the tray or pallet through the entire process that involves multiple tasks at multiple modules. The details of this discussion are provided in Appendix A, together with illustrations of some of the models obtained this way. However, in order to enable process mining on the full system, we have to conceptualize how the modules interact, or more specifically how the objects in the system flow between the modules throughout the entire process, with respect to a pallet, a tray, or a combination of both. Thus, we opt to create models that cover the entire process for a tray or pallet.

To do so, we approached the problem using so-called lifecycle models, which integrate the behavior of an object in different modules. The traces of such a model are concatenations of occurrences of an object that participates in multiple tasks. Thus, it describes the lifecycle of an object in the system along the various tasks, represented by the messages sent during the execution of these tasks, hence the name lifecycle model. We base the models on the events of the messages sent between the modules. These messages connect the different parts, and exactly these messages are logged as events like we mentioned in Chapter 3. Moreover, this will directly help us with extracting event logs later in Chapter 5, as we will further elaborate throughout this chapter. Similar (though more extensive) lifecycle models were made for the specific Vanderlande system. We concluded with experts that these models are useful for understanding flows and figuring out what we need to extract to analyze the use cases they initially defined for us (inbound, de-stacking, stacking, in-to-out) and how to analyze the physical flow.

The remainder of this chapter is broken down into a few sections. First, we provide the concepts of the introduced lifecycle models in Section 4.2. Then we present a lifecycle model for the tray in Section 4.3, after which we present the lifecycle models of a pallet in Section 4.4. In both sections we identify the lifecycle start and end points for each of the analysis use cases. Afterwards, we present the lifecycle model of the tray and pallet together in Section 4.5. Here we first introduce some more concepts to be able to distinguish the pallet and tray in the process, and then focus on the interaction between pallet and tray. For all lifecycle models in Sections 4.3, 4.4, and 4.5 we explain how to read them and elaborate on the flow of the process. We conclude the chapter in Section 4.6 where we round up the conceptualization of warehouse automation systems. Finally, we shortly elaborate on the remainder of the thesis.

4.2. **Lifecycle Model Concepts**

The dynamics of the interplay between a pallet and a tray are complex, and applying BPMN or Petri nets with precise semantics yields complex models like we shortly mentioned in Section 4.1 (see Appendix A for more details). The models we want do not need to have precise operational semantics, but need to capture the general sequencing of steps. Hence, we use a more informal modeling notation. Before diving straight into the models we first introduce some modelling concepts. The legend in Figure 16 shows the basic concepts of the modelling notation that we
introduce. In Section 4.5 we discuss the lifecycle interactions, where we will introduce further notation concepts. Any extra text that is on/in the model is purely there for further explanation and is intended to be self-explanatory. We created these models by reading documentation, asking domain experts, conceptualizing, and iterating over them by discussing with modeling experts and domain experts. The final models were verified both as ‘good to understand’ and ‘valid models of the system’ together with these experts. Note that the models presented below are abstractions from the real models, just as the examples in Chapter 3 were abstractions from the real system.

Activities
Activities are shown as rounded rectangles, either white or gray (with a double outline), and represent the (events of the) messages being sent from and to the modules. The coloring indicates that the activity is about the task status (i.e. Assign, Start, Complete), whereas the white (or no) coloring indicates that the activity is either a location or a module status update. The double outline on a gray activity indicates that the message was sent by the conductor. Each activity is annotated with the corresponding process step from Figure 8.

Hence, executing an activity corresponds to the sending of a message as described in Chapter 3, and thus the creation of a log entry for this message describing which message it was and the corresponding Object ID, Task ID, and timestamp. Because each activity relates to the logged events like this, we can later use the lifecycle model to collect all the necessary information to build an event log that describes the lifecycle. Specifically, the lifecycle model will help with identifying the start and end event for a trace and a case identifier, which are the main inputs to the log extraction specification we will define in Chapter 5.

Choice Squares / XOR Operators
Two or more separate edges coming out of or going into a single activity indicate a XOR-split and XOR-join respectively. The small grey squares with a number indicate choices in the flow of the model, so they are technically XOR-splits. We do not provide a formal specification when which choice is taken, but the label helps the reader follow the flow of a particular process or dynamic. Moreover, the numbering corresponds to the process steps in Figure 8. When the choice has been made, the particular corresponding number has to be kept in mind as some activities can be labeled with a number. This number indicates that the activity only occurs if the previous choice split was that exact number, otherwise the activity is skipped [36].

Places
Circles in the model, either with a dashed line or not, indicate a ‘waiting place’, and correspond to a tray or pallet reaching a storage location (see Figure 10 in Chapter 3). The dashed line indicates that the object is not tracked through its identifier, meaning it momentarily ‘disappears’ in the process, only to re-appear when the process continues.
Figure 16: System lifecycle model of a tray, annotated with the process steps of Figure 8
4.3. System Lifecycle Model of a Tray

The system lifecycle model of a tray is presented in Figure 16. As the full lifecycle of a tray does not have an end, as the ID is persistent, the lifecycle model is cyclic. However, depending on the use case, there are generally two start ‘states’ to be considered, both of which have been depicted as a place in the model. To have an as-complete-as-possible explanation of the model we consider a case where an empty tray is filled at de-stacking, then goes to storage, and is finally used at the stacker, after which it goes to the ‘empty tray buffer’ where it waits to be filled again. The process is then as follows, with the corresponding activities and places in bold:

- The tray starts at the Empty Tray Buffer.
- When it reaches the entry of the de-stacker a location status message is sent (At Stacker Entry).
- The following Transport Task Complete does not occur as we have not chosen 4 before.
- The De-Stack Task is then Assigned and Started, after which the tray is filled (Put in Tray).
- The tray leaves the de-stacker (At De-Stacker Exit), and the task is completed (De-Stack Task Complete).
- A Transport Task to go towards storage is Assigned and Started.
- Once the tray is at Storage Entry the Transport Task is Completed and a Store Task is Assigned and Started.
- A status message (Storing) and finally the storage location are sent (At Storage Location), after which the Store Task is Completed.
- The tray now resides in storage, as we illustrate by the ‘In Storage’ place. Here it waits for a while until it is required for a stacking task.
- When so, a Retrieve Task is Assigned and Started.
- Again a ‘busy’ message is sent (Retrieving) and later when At Storage Exit a location update is sent as well.
- The Retrieve Task is Completed, and a Transport Task is Assigned and Started.
- The tray is being transported (Transporting), and now a choice must be made for 4, 7, or 6. As we want the tray to go towards the stacker we choose 7.
- Once the tray is at Stacker Entry the Transport Task is Completed and a Stacking Task is Assigned and Started.
- Now, the stacker takes one or more products from the Tray, indicated by the self-loop, after which the tray exits the stacker (At Stacker Exit).
- The Stack Task is Completed and again we reach a choice. If the tray is not empty we can only go towards 6, if it is empty it can go towards either empty or 6.
- Previously it was decided that the tray would return to the Empty Tray Buffer, hence we choose empty and the tray process ‘ends’ in the Empty Tray Buffer.

Process Lifecycle Model Definitions

To analyze the various use cases defined by the engineers we need to obtain the lifecycle of the tray for each use case (inbound, de-stacking, stacking, in-to-out). From the system lifecycle model of a tray (Figure 16) we can derive multiple process lifecycle models of a tray, which describe how a tray behaves in a particular use case. We can obtain these process lifecycle models by projection onto a subset of the nodes and edges of the system lifecycle model. Moreover, for each use case we can use the corresponding process steps to do so, as the system process model has been labelled with corresponding process steps (from Figure 8). Below, for each use case, we will define the start and end activity (or place), and intermediate choices in sequence for the projection to obtain the process lifecycle model. Note that the ‘inbound’ use case is not mentioned here as this does not involve trays.

- De-Stacking: (projection on process steps 4, (5a), 5b, and 6)
  - Start: Empty Tray Buffer or In Storage (an empty tray)
  - End: In Storage (now a full tray)
  - Choices: none or 4 (depending on the start respectively)

- Stacking: (projection on process steps 6 and 7)
  - Start: In Storage (a full tray)
  - End: Empty Storage Buffer or In Storage (depending on choice)
  - Choices: 7 \( \rightarrow \) empty or 6

- In-to-Out: (= system lifecycle model)
  - Start: Empty Tray Buffer or In Storage (an empty tray)
  - End: Empty Tray Buffer or In Storage (an empty or semi-full tray)
  - Choices: 4 or none (depending on the start respectively) \( \rightarrow \) 7 \( \rightarrow \) empty or 6

We will later use these projected process lifecycle models, especially the start and end events, to scope log extraction.
4.4. System Lifecycle Models of a Pallet

Figure 17 illustrates both the lifecycle model of the inbound pallets and that of outbound pallets. Both models are very sequential and have a specific start and end state because the pallets have a unique id per process execution as they are unique physical pallets – in contrast to a tray. The end state of both models is indicated as a place with a double line.

Outbound Pallet
The outbound pallet model is very short, as transporting the pallets to the dock and loading them on the truck is out of scope as we stated in Chapter 3; thus we do not model it. We explain the process below, again with the activities and places in bold.

- The process starts when a Stack Task is Assigned and Started.
- Products are stacked on the empty pallet one by one, as indicated by the self-loop on Put on Pallet.
- Once the Stack Task is Completed the pallet is automatically brought to the exit of the stacker (At Stacker Exit), where it Awaits Pickup.

Inbound Pallet
The inbound pallet model is a lot longer, but still very comprehensive due to its sequentiality. Again, we list the process steps and highlight the activities and places in bold.

- The process starts when a pallet is received (Receiving), after which it is immediately Assigned a Transport Task.
- The Transport Task is Started and the pallet is brought to storage (Transporting).
- Once the pallet is at Storage Entry the Transport Task is Completed and a Store Task is Assigned and Started.
- A status message is sent (Storing) and when it reaches the storage location a location message is sent (At Storage Location), after which the Store Task is Completed.
- Just as with the tray, the pallet now resides in Storage. Here it can stay for quite long, as the receiving is supposed to happen a good while ahead of time before products are needed, i.e. there needs to be sufficient stock in the warehouse at all times.
- When the products on the pallet are required for stacking the pallet is retrieved and transported to the de-stacker, where the de-stack task is started. (Retrieve Task Assign until De-Stack Start)
- Here layers of products are removed one by one until the pallet is completely empty (Remove Layer).
- Once the De-Stack Task is Completed, the empty pallet is automatically transported out of the system, where a final location message is sent to indicate the exit of the pallet (Empty Pallet Exit), which also indicates the end of the lifecycle of the inbound pallet.

Process Lifecycle Model Definitions
Just as with the tray system lifecycle model in the previous section, to analyze the various use cases (and to scope log extraction) we define a projection using the process steps of each respective use case to obtain the process lifecycle models for the pallet(s). Again, per use case, we define the start and end activity (or place), and no choices as there are none here. Note that for the In-to-Out process we are technically looking at 2 pallets, the inbound pallet and the outbound pallet. As such the In-to-Out use case has two ‘halves’, as also indicated by the dashed line in Figure 17, and thus also has two pallet identifiers.

- Inbound: (projection on process steps 2 and 3)
  - Start: Receiving
  - End: In Storage
- De-Stacking: (projection on process steps 4 and 5a (and 5b))
  - Start: In Storage
  - End: Empty Pallet Exit
- Stacking: (projection on process step 7 (and 6))
  - Start: Empty Pallet for Stacking
  - End: Awaits Pickup
- In-to-Out: (= system lifecycle model)
  - Inbound use case start, de-stacking end
  - Stacking use case
Figure 17: Inbound (left) and outbound (right) pallet system lifecycle model, annotated with the process steps of Figure 8.
4.5. Complete System Lifecycle Model

The use cases De-Stacking, Stacking, and In-to-Out all involve both a pallet and a tray. Analyzing these use cases involves analyzing the interplay between pallets and trays, and thus we need a combined model. We do not use an artifact-centric model to model this interplay of the objects as we discussed in Chapter 2, like X. Lu [11] did for ERP systems. We saw in Chapter 3 that the data model is not fully relational (i.e. we do not have a key or foreign keys to link tables together), and the artifact-centric process mining framework XTract2 is not available. Hence, we decide to combine the earlier presented system lifecycle models of a tray and a pallet into one complete system lifecycle model. In Figure 19 we present this lifecycle model. The model uses the exact same concepts as before, with the addition of a few concepts as follows:

- An **AND-split** is portrayed by one arrow that splits up into two or more, thus having only a single connection point to the activity.
- An **AND-join** is portrayed by two arrows joining on *each other* before reaching the activity, hence also only having a single connection point to the activity they join on.
- Pallet activities are colored blue to distinguish them from tray activities. The one activity with a gradient indicates that a message for both the pallet and the tray is sent when they arrive, thus there are two messages in parallel here; this also means that there are 2 events logged.
- The next concept introduces how other ‘duplicate’ messages are handled: instead of putting two activities after each other or having a self-loop, a specific number of messages is indicated by a blue colored number between brackets, in this case: (2). There are 2 messages here because a task is completed for the pallet as well as for the tray if the tray came from storage. If the tray came from the empty tray storage there is only 1 Transport Task Complete activity here. This also means that 2 events are logged.
- A sub-process is added in the form of a blue rounded rectangle with a smaller rectangle in it: the Inbound Pallet Process. As the first half of this process (bringing the pallet to storage) is disconnected in time from the rest of the process, this is separated out to save on space and complexity. Moreover, as there are no interactions between objects in this part of the process yet, it can safely be left out.

The process follows the same flows as the two models described above. The interesting parts are the interactions between pallet and tray, i.e. the handover of products at de-stacking and stacking. Thus we explain both in detail again, as now a look into the complete process is given instead of only a look into 1 tray or pallet, and thus the interplay between the two objects must be explained.

**De-Stacking**

Once both the pallet and a set of trays are ready (i.e. are at the de-stacker entry) the de-stacking can be started. The first layer needs to be removed from the pallet before products can be put into trays, which is modeled as a separate activity, which did not exist earlier. After the first layer is removed the flow splits in two parallel flows: 1 for the tray(s), and 1 for the pallet. Hence, trays are filled while the pallet is being emptied concurrently. After the pallet is empty and all trays are filled the task is completed. The empty pallet then exits the system in parallel with the tray(s) being transported towards storage.

Note that for the relation between 1 pallet and 1 tray a (partial) ordering can be created as in Figure 9, but for the relation between multiple trays and a pallet there is no one specific flow of messages. Thus, the self-loops on ‘Remove Layer’ and ‘Put in Tray’ + ‘At De-Stacker Exit’ can happen at any moment in time relative to each other. Moreover, when the pallet is empty it will be transported out of the de-stacker immediately, while some trays still might need to be filled. Hence, the ‘Empty Pallet Exit’ can in reality also occur at any moment after the last ‘Remove Layer’ message has been sent. Note that the pallet lifecycle ends in a specific ‘end’ place, whereas the tray lifecycle ends in the ‘In Storage’ or ‘Empty Tray Storage’ place.

This interplay between pallet and tray leads to a lot of different possibilities for message flows, as we illustrate with an example in Figure 18. If there are 4 messages about the pallet and 5 about the tray, since they physically do not interfere with each other the interplay of these messages can result in many possible message sequences (permutations). After each tray event any subset of the four pallet events can happen, and the remaining subsets can happen in any order as well (as long as the separate ordering of pallet and tray is kept intact). The figure shows only the edges from the tray events to the pallet events in red (and not vice versa), indicating all possible flows from tray event to pallet event in a trace. This already results in 20 additional edges (5x4). The challenges this poses will become more clear in Chapter 6 when we analyze how existing process mining tools visualize this concurrent behavior.
Stacking
The interplay between pallet and trays at stacking is a lot less intricate, and the implications have already been covered at de-stacking. Trays arrive one by one and the products they hold are put onto the pallet one by one. The only challenging aspect is the exit of trays. As each tray will come at a different time the exit of the tray can be before or after the task is completed and before or after the stacked pallet exits the stacker. This again leads to many different options for the flow, exactly as illustrated in Figure 18. Though, it must be noted that this part of the model has less parallel activities, and therefore there will be less permutations.

Process Lifecycle Model Definitions
Finally, we again present the process lifecycle model definitions per use case, to later help us with defining a scope for the event log extraction. Moreover, this time we present the definitions per object and use case, as we now have two objects. Recall the process steps and involved objects for each of the four use cases:

- Inbound: step 2 and 3 (pallet)
- De-Stacking: step 4, 5ab, and 6 (pallet, tray)
- Stacking: step 6 and 7 (tray, pallet)
- In-to-Out: step 2, 3, 4, 5ab, 6, and 7 (pallet, tray, pallet)

Hence, projecting on each of these steps yields the following lifecycles per object and use case; again listing the start and end activity (or place) for each object, and specifying the choices to follow in sequence – if any.

- Inbound
  - Pallet Start: Receiving
  - Pallet End: In Storage

- De-Stacking
  - Tray Start: Empty Tray Buffer or In Storage (an empty tray)
  - Tray End: In Storage (now a full tray)
  - Choices: none or 4 (depending on the start respectively)
  - Pallet Start: In Storage (inside sub-process in Figure 19)
  - Pallet End: Empty Pallet Exit

- Stacking
  - Tray Start: In Storage (a full tray)
  - Tray End: Empty Storage Buffer or In Storage (depending on choice)
  - Choices: 7 → empty or 6
  - Pallet Start: Empty Pallet for Stacking (implied in Figure 19; thus we refer to Figure 17)
  - Pallet End: Awaits Pickup

- In-to-Out (= system lifecycle model)
  - Tray Start: Empty Tray Buffer or In Storage (an empty tray)
  - Tray End: Empty Tray Buffer or In Storage (an empty or semi-full tray)
  - Choices: 4 or none (depending on the start respectively) → 7 → empty or 6
  - Inbound Pallet Start: Inbound use case start
  - Inbound Pallet End: De-stacking use case end
  - Stacking/Outbound Pallet Start+End: Stacking use case
Figure 19: Complete system lifecycle model for pallet & tray combined, annotated with process steps from Figure 8
4.6. Concluding the Conceptualization of Warehouse Automation Systems

In the previous sections, we set up a baseline to work with for the following chapters. We provided conceptual models and process descriptions of warehouse automation systems and described how the objects interact. Moreover, we introduced the specific start and end points for the lifecycles of the objects in the system for each of the use cases we defined in Chapter 3. Thus, we can answer research question 1:

(I) Given a material handling system with automated batching, including multiple different case identifiers, we would like to gain conceptual understanding of the processes and the difficulties it brings for log extraction.

We have gained conceptual understanding by drafting various models: a high-level overview of the steps in a warehouse, a blueprint of a warehouse, an example execution of the process, two message sequence diagrams, and analysis use cases in Chapter 3 and lifecycle models in this chapter. With all these models and the corresponding explanations we have a complete view of the warehouse automation system. Thus, we define the challenges we face for process mining for these systems as follows:

- How to define a case identifier from the multiple identifiers (Tray IDs, Pallet IDs, Task IDs)?
- How to extract the specified lifecycles of objects from the data?
- How do we make the connections between various Tasks and objects as we modeled, and extract this from the data?

Answering these three questions should lead to the setup of a log extraction approach. We can use all defined lifecycle models to help us set up an ETL approach and to identify which messages/events need to be included in the log extraction specification (LES). In the LES we need to specify which events belong to a view and how to scope that view based on the projected lifecycle models for each use case.

The following chapters will go over the steps we took to do analysis on the warehousing system. First, in Chapter 5, we explain the log extraction approach by answering the above three questions and we conceptualize the actual approach and queries for use in more (material handling) systems. We end the chapter by briefly stating the results, i.e. some statistics on the extracted event logs from the system in our case study. Immediately thereafter the pre-processing and analysis of the logs is discussed in Chapter 6. In Chapter 7 we do a combinatorial study and systematically evaluate the various pre-processing approaches and resulting models. Afterwards, we describe a methodology for applying process mining in material handling systems in Chapter 8. Finally, we conclude the thesis in Chapter 9 by a discussion of future work and the limitations of this work, where we define open research problems and questions.
5. EVENT LOG EXTRACTION

In the previous two chapters we achieved an understanding at a conceptual level of warehouse automation systems. More specifically, in Chapter 3 we developed an understanding of the system and confirmed existence of multiple objects in 1:n relations that are moved by system components regulated through tasks. These tasks are coordinated by message exchanges between system components (modules) and a conductor (super module). Events are logged that detail the type of message, the ID of the task and object (Task ID, Pallet ID, Tray ID), and a timestamp. Each of the different messages is logged in separate tables. Furthermore, we identified analysis use cases that require us to consider events from multiple tables, as they include multiple objects and various messages. On top of that, the trays keep their identifiers throughout the entire process, and hence we cannot use the Tray ID to distinguish different process executions. Thus, event log extraction is non-trivial and requires scoping events. To help us do so, in Chapter 4, we developed conceptual system lifecycle models, which model how trays and pallets move through the system in the context of tasks and the corresponding message exchanges. Projecting these system lifecycle models onto tasks (process steps) relevant for an analysis use case yields the set of all events (and related identifiers) that need to be considered together to extract an event log for that use case. Given the gained understanding we formulated three questions:

- How to define a case identifier from the multiple identifiers (Tray IDs, Pallet IDs, Task IDs)?
- How to extract the specified lifecycles of objects from the data?
- How do we make the connections between various Tasks and objects as we modeled, and extract this from the data?

Thus, in this chapter we aim to answer these three questions by formulating an approach for event log extraction. The event logs we want to extract have to consider both pallets and trays. As such, we want to analyze any object and therefore any case identifier from the data. Moreover, we want to be able to extract event logs that correspond to a single use case, or a combination of a set of use cases, as we defined in Chapters 3 and 4. This allows us to analyze the system as a whole and to investigate the interaction between pallets and trays but also allows for analyzing smaller sub-processes and potentially separate modules.

The ideal solution is an event type/log extraction specification as by E. Nooijen and X. Lu in their work, which specifies source data for each event and defines case identifiers. This event type/log extraction specification can be obtained semi-automatically or fully automatically for proper relational data [34] [11]. However, next to the tooling not being publicly available, we cannot directly apply this approach as we do not have full relational data, but rather a number of individual tables per message from the modules, which together describe the process and tasks. Moreover, we found that IDs are being reused (recall the Tray ID), and thus we need to introduce case time-frames by specifying the start and end times for a trace, i.e. projecting the lifecycle of an object onto the data. Note that we will not extract one large event log that can handle all use cases, but rather we decide to extract multiple event logs, i.e. views on the data.

Hence, we propose the following approach based on the conceptual models of Chapter 3 and 4, in which we derive queries for log extraction through a manual implementation of an ETL approach, in three general steps:

1) Define which use case – sub-process(es) – we want to extract, to be done per analysis use case
2) Specify boundaries of the lifecycles of the objects to be included in the log, and extract them by help of the lifecycle models from Chapter 4
3) Extract logs automatically using a generic query based on the parameters from Step 2

We achieve log extraction in three steps, as is illustrated in Figure 20. The ‘papers’ represent documents or objects, which are input or output to or from the steps, which are illustrated as rounded rectangles. The first input is the obtained domain knowledge, specifically the lifecycle models we developed in Chapter 4 (which answers question 1 above). In Section 5.1 we gather requirements for the log extraction and formulate a log extraction specification (LES). In Section 5.2 we explain an approach for extracting the LES from the data. Then in Section 5.3 we explain how to extract event logs using the LES as input and discuss the various options for the case identifier. Finally, in Section 5.4 we shortly discuss the approach and present a few event logs that we extracted from the case study system, which we will then further analyze in Chapter 6.
5.1. Requirements for Log Extraction

Based on the literature on artifact-centric process modeling and other work on multiple case identifiers, the lifecycle models we defined in Chapter 4, the multiple case IDs and persistent IDs found in Chapter 3, taking one step back and looking at existing extraction approaches, to extract an event log we need to do the following:

A) Pick a case identifier
B) Scope which events to include
C) Specify time framing to deal with persistent identifiers (tray ID)

We first shortly discuss the source data a bit more by showing example data for the running example in Section 5.1.1. Afterwards, in Section 5.1.2, we discuss how or what to pick as a case identifier (A), and we show naïve log extraction on the running example and discuss the forthcoming issues. From that we discuss how to scope which events to include (B) by using the lifecycle models from Chapter 4 on the running example in Section 5.1.3. Then, we discuss how to deal with persistent identifiers by specifying time-frames (C) – again by illustrating this on the running example. We conclude in Section 5.1.4 by specifying and explaining the Log Extraction Specification, including the LES for the example.

5.1.1. Source Data

In Chapter 3 we explained the data model for logging on a high level, i.e. per message we have an object ID, Task ID, and timestamp, and each of these messages is stored in its own table. To explain log extraction from this kind of data, we show example tables for (some of) the events of the running example of Figure 22 in Figure 21. The coloring in the tables indicate to which Stacking Order each event belongs, in line with the coloring in Figure 22. Each table holds the object ID, a timestamp, and the corresponding Task ID. The start events correspond to the first occurrence of an object, and for Ta also to the event on the same level as P2 and SO2 (and the first one immediately beneath that for Tb). The end events all correspond to the “End of life-cycle” boxes in Figure 22.

As we mentioned in the introduction we do not have a fully relational data model with all references made explicitly. Therefore, we do not have any keys or identifiers for the tables, like is customary for most database models. Thus, we have to define what can or should be linked. In general, links can be made using the Object ID and Task ID. However, for the Tray ID this has to be done with extra care as the lifecycle is indefinite and thus linking together events based on the Tray ID would result in one large case, instead of multiple cases as defined by the process lifecycle model definitions of a tray in Chapter 4. We elaborate how we find these start and end events for an object and link them together to obtain a case time-frame in the following sections. Moreover, we explain why a case time-frame is necessary by shortly discussing and illustrating what happens when we do not use this. However, we first need to specify a case ID to define our case before we can scope it in a time-frame.
In earlier works on artifact-centric log extraction [11] [34], log extraction first defined the notion of a case by defining the case identifier column, and all the different timestamp columns. All records in the timestamp columns that can be associated to the same records in the case ID column belong to the same case. E. Nooijen also showed that the case ID record can be defined by multiple case ID columns and corresponding timestamp columns [34]. Technically, a user (or an algorithm) first specifies these case ID and timestamp columns, and then a query/algorithm having case ID and timestamp columns as parameters extracts the events and groups them into traces automatically.

Following this exact approach, we want to define a case identifier that consists of multiple case ID columns. Which columns these are depend on the use case we want to analyze. In general, we select one or more ObjectID columns from the data. To combine the various identifiers we follow the approach opted by J. Buijs [32], where we take a ‘lower level’ identifier. With respect to the amount of products, that is in or on an object, the finest granularity – and hence ‘lowest level’ – is when the products reside in a tray. Hence, for now, we decide to select the Tray ID as the lowest level identifier, and simply add the other identifiers (Pallet ID(s)) by concatenating them. We will elaborate on other possibilities for case identifiers in Section 5.3.2, after we have fully explained the log extraction approach.
To obtain an event log that includes all events for the case (i.e. also includes the pallet events preceding a particular tray), we need to enrich the traces of the tray by adding the corresponding pallet events and tasks. More specifically, for the defined case, we need to extract events of each object that is in the case identifier.

Applying the above ideas of selecting multiple case ID columns and enriching the traces on the example of Figure 21 and Figure 22 with a case identifier Pallet ID-Tray ID-StackingPallet ID for Ta and Shop Order 1 would result in the following (simplified) log:

**Table 5.1: Simplified event log for Ta, based on P1-Ta-SO1**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Case ID</th>
<th>Timestamp</th>
<th>Object ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-stack</td>
<td>P1-Ta-SO1</td>
<td>12:24</td>
<td>P1</td>
<td>DST1</td>
</tr>
<tr>
<td>Put in Tray</td>
<td>P1-Ta-SO1</td>
<td>12:42</td>
<td>Tb</td>
<td>DST1</td>
</tr>
<tr>
<td>Empty Pallet Exit</td>
<td>P1-Ta-SO1</td>
<td>12:47</td>
<td>Tb</td>
<td>DST1</td>
</tr>
<tr>
<td>Put on Pallet</td>
<td>P1-Ta-SO1</td>
<td>13:01</td>
<td>Ta</td>
<td>ST1</td>
</tr>
<tr>
<td>Put on Pallet</td>
<td>P1-Ta-SO1</td>
<td>13:03</td>
<td>Ta</td>
<td>ST1</td>
</tr>
<tr>
<td>Stacking Pallet Exit</td>
<td>P1-Ta-SO1</td>
<td>13:10</td>
<td>SO1</td>
<td>ST1</td>
</tr>
<tr>
<td>Put on Pallet</td>
<td>P1-Ta-SO1</td>
<td>13:30</td>
<td>Ta</td>
<td>ST2</td>
</tr>
</tbody>
</table>

We observe that our event log contains events that are not relevant for this case. The highlighted rows of Table 5.1 show the filling and emptying of Tb, and Ta being used for stacking in SO2. Since we only want to analyze SO1 and Ta, having these events in our trace/event log poses an issue. Note that this event log leaves out all the transportation and storage that happens in between de-stacking and stacking, as explained in Chapters 3 and 4, which would add more unrelated and irrelevant events to the event log. Thus, we cannot simply take all events that are related to either Ta or SO1. Taking all events related to SO1 gives us the unwanted events of Tb, whereas taking all events related to Ta gives us the event of Ta being used in the second stacking order.

Moreover, if we want to consider a specific use case we need to further scope which events we extract. For instance, if we wanted to only look at De-Stacking all the events related to stacking (i.e. ‘Put on Pallet’ onwards) are irrelevant and should not have been extracted. Hence, we have to scope which events we extract for a (use) case, using our lifecycle models as an input. We will see that this then also requires defining and using the start and end time of the lifecycle of the object.

### 5.1.3. Scoping and Case Time-frames

Deciding which events we need to include in the event extraction is a choice to be made by the help of the lifecycle models we presented in Chapter 4. More specifically, using the process lifecycle models we know exactly where (i.e. with which event) each use case starts and ends. For instance, recall the de-stacking use case process lifecycle definition:

- Tray Start: Empty Tray Buffer or In Storage (an empty tray)
- Tray End: In Storage (now a full tray)
- Choices: none or 4 (depending on the start respectively)
- Pallet Start: In Storage
- Pallet End: Empty Pallet Exit

We have obtained these start and end times by projecting the process steps we defined in Chapter 3 onto the full system lifecycle models. Now, to find which events we need to include, and thereby which tables we need to query on, we can simply take the set of all events in between (and including) the start and end time for each of the objects. Applying this scoping onto the log extraction of Tb*SO1, specifically leaving out Ta, for the De-Stacking use case, we obtain the following log:

**Table 5.2: Event log for De-Stacking use case for Tb (empty pallet exit for P1 left out for brevity)**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Case ID</th>
<th>Timestamp</th>
<th>Object ID</th>
<th>Task ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-stack</td>
<td>P1-Tb-SO1</td>
<td>12:24</td>
<td>P1</td>
<td>DST1</td>
</tr>
<tr>
<td>Put in Tray</td>
<td>P1-Tb-SO1</td>
<td>12:42</td>
<td>Tb</td>
<td>DST1</td>
</tr>
<tr>
<td>In Storage</td>
<td>P1-Tb-SO1</td>
<td>13:06</td>
<td>Tb</td>
<td>DST1</td>
</tr>
<tr>
<td>Put in Tray</td>
<td>P1-Tb-SO1</td>
<td>13:17</td>
<td>Tb</td>
<td>DST2</td>
</tr>
<tr>
<td>In Storage</td>
<td>P1-Tb-SO1</td>
<td>13:40</td>
<td>Tb</td>
<td>DST2</td>
</tr>
</tbody>
</table>
Again highlighted in red and *italics*, we observe that we extract *more* than we wanted or specified. This is because Tb participates in *two* de-stacking tasks in our data and the Tray ID of Tb is persistent through the entire dataset. Hence, even if we *scope on a subset of events* we do not necessarily get a log that includes *only* what we intended to extract (i.e. only the first three rows of Table 5.2). Thus, next to specifying which events we need to extract, we also need to specify an *end time* until which we need to extract this set of events.

Now, consider the same log from Table 5.2 again, but this time we want to extract the de-stacking of P2 and Tb – i.e. the last 2 rows and a different de-stack event. In this case, if we do not also specify a *start time* we will again extract the de-stacking from P1 onto Tb. Hence, a trace must be ‘framed’ by a *start time and end time*. Thus, we introduce *case time-frames* to encapsulate the object lifecycle by start and end time in each respective trace.

To find the start and end times for the time-frame of an object we must identify the start and end of the (use) case, which we have done in Chapter 4 by specifying *process lifecycle models*. We can then find the start and end times in the ‘timestamp’ column of the corresponding event type table. Thus, using the process lifecycle models and the data model we can quite easily find the start and end times for each use case and object. However, there are exceptions to this. For instance, for the In-to-Out use case we have *two* end points for a tray: either in storage or in the empty tray storage. If both events exist for this tray we can use logic to determine which one belongs to this lifecycle. Given the *timestamp of the event of the stacking task* the tray participated in we check which of the two potential end events has a larger timestamp, as it must have happened after the stacking. If both have a larger timestamp, we compare the two events to each other, where the smallest timestamp is the ‘winner’, i.e. the smallest timestamp of the two events is the end of the lifecycle. This holds true because the larger timestamp would indicate that the tray participated in another ‘round’ in the process, and hence that event belongs to another case. Thus, using this logic we can find the *end time* of the case for this particular use case for a tray. For the two potential start events of the in-to-Out use case (and any other use case that has two start or end events), we can use similar reasoning to obtain the correct start event. In Section 5.2 we illustrate how to do this in the LES extraction as well.

Using similar reasoning, and the other lifecycle definitions from Chapter 4, we can find the corresponding start and end events (and hence times) for each of the objects for each of the use cases. Thus, we now have a way to determine the case time-frame for a trace. This process is a very *manual task* and requires a lot of knowledge of the system at hand, which is why we first drafted the conceptual models in Chapter 3 and 4.

### 5.1.4. Log Extraction Specification

Combining the results of the previous sections, we obtain a log extraction specification, which is illustrated in Figure 23. The log extraction specification defines *what to extract for each case*. Moreover, it explicitly shows the object IDs and the case time-frames for these objects, as such bounding the lifecycle of each object to the case, corresponding to the use case we analyze. The LES for the total event log is simply a list or table of the defined cases, like in Figure 24, which shows 4 cases as defined in Figure 23: 1 per row. We will elaborate how to obtain this in Section 5.2. The LES thus holds a case definition per row, which includes *all objects belonging to that case and the timeframes of each of those objects*. The LES does not specify the set of events to extract during the automated event log extraction. This will be a *separate input* that is more statically derived from the domain knowledge, by looking at the lifecycle models as we shortly described in the previous section.

![Figure 23: The Log Extraction Specification per case](image-url)
The input for this LES is purely domain knowledge (mainly the lifecycle models) as we elaborated for the case timeframes per object. We link objects together based on Object ID, Task ID, and sometimes need to base decisions on the timestamps of the events as there can be multiple occurrences of an event for an Object ID. The resulting decision is often the first or earliest event that happens with respect to the event we are looking at for an object, like we discussed in Section 5.1.3. Hence, we now have a way of manually obtaining a LES. However, doing so for a large number of cases would be unreasonable as it requires an even larger amount of work. Therefore, we want to be able to extract these LESs (semi)automatically by defining a query/algorithm to do so.

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-1</th>
<th>End-1</th>
<th>De-stack Task ID</th>
<th>ID2</th>
<th>Start-2</th>
<th>End-2</th>
<th>Stack Task ID</th>
<th>ID3</th>
<th>Start-3</th>
<th>End-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Ta</td>
<td>12:42</td>
<td>13:05</td>
<td>ST1</td>
<td>S01</td>
<td>13:00</td>
<td>13:10</td>
</tr>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Tb</td>
<td>12:42</td>
<td>13:06</td>
<td>ST1</td>
<td>S01</td>
<td>13:00</td>
<td>13:10</td>
</tr>
</tbody>
</table>

Figure 24: Obtained LES for running example (showing 4 cases, 1 per row)

Using the LES in Figure 24 for the example source data of Figure 21 and Figure 22 would then lead to the correctly extracted event log of Figure 25 (for case P1-Ta-SO1). Next, we explain how to design a query that automatically obtains the LES of Figure 24 from the source data of Figure 21.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Activity</th>
<th>CaseID</th>
<th>Location/Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>12:24</td>
<td>Transport Task Assign</td>
<td>P1-Ta-SO1</td>
<td>Pallet Entry</td>
</tr>
<tr>
<td>12:24</td>
<td>Transport Task Start</td>
<td>P1-Ta-SO1</td>
<td>Pallet Entry</td>
</tr>
<tr>
<td>12:26</td>
<td>At Storage Entry</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>12:26</td>
<td>Transport Task Complete</td>
<td>P1-Ta-SO1</td>
<td>At Storage Entry</td>
</tr>
<tr>
<td></td>
<td>. . . Further transport and storing and retrieving of pallet . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:35</td>
<td>De-Stack Task Assign</td>
<td>P1-Ta-SO1</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:35</td>
<td>De-Stack Task Start</td>
<td>P1-Ta-SO1</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:35</td>
<td>At De-Stacker Entry</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>12:36</td>
<td>Remove first layer</td>
<td>P1-Ta-SO1</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:41</td>
<td>At De-Stacker Entry</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>12:42</td>
<td>Put in Tray</td>
<td>P1-Ta-SO1</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:43</td>
<td>At De-Stacker Exit</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>12:43</td>
<td>De-Stack Task Complete</td>
<td>P1-Ta-SO1</td>
<td>De-Stacker</td>
</tr>
<tr>
<td>12:43</td>
<td>Transport Task Assign</td>
<td>P1-Ta-SO1</td>
<td>Storage</td>
</tr>
<tr>
<td>12:43</td>
<td>Transport Task Start</td>
<td>P1-Ta-SO1</td>
<td>At De-Stacker Exit</td>
</tr>
<tr>
<td></td>
<td>. . . Further transport and storing of tray . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12:47</td>
<td>Empty Pallet Exit</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:00</td>
<td>Stack Task Assign</td>
<td>P1-Ta-SO1</td>
<td>Stored</td>
</tr>
<tr>
<td>13:00</td>
<td>Retrieve Task Assign</td>
<td>P1-Ta-SO1</td>
<td>Stored</td>
</tr>
<tr>
<td></td>
<td>. . . Further retrieval and transport of tray . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:01</td>
<td>At Stacker Entry</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>13:01</td>
<td>Transport Task Complete</td>
<td>P1-Ta-SO1</td>
<td>At Stacker Entry</td>
</tr>
<tr>
<td>13:01</td>
<td>Stack Task Start</td>
<td>P1-Ta-SO1</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:02</td>
<td>Take from Tray</td>
<td>P1-Ta-SO1</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:03</td>
<td>Put on Pallet</td>
<td>P1-Ta-SO1</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:03</td>
<td>At Stacker Exit</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
<tr>
<td>13:03</td>
<td>Store Task Assign</td>
<td>P1-Ta-SO1</td>
<td>At Stacker Exit</td>
</tr>
<tr>
<td></td>
<td>. . . Further storing of tray . . .</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:05</td>
<td>Put on Pallet</td>
<td>P1-Ta-SO1</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:05</td>
<td>Store Task Complete</td>
<td>P1-Ta-SO1</td>
<td>Stored</td>
</tr>
<tr>
<td>13:06</td>
<td>Stack Task Complete</td>
<td>P1-Ta-SO1</td>
<td>Stacker</td>
</tr>
<tr>
<td>13:10</td>
<td>At Stacker Exit</td>
<td>P1-Ta-SO1</td>
<td></td>
</tr>
</tbody>
</table>

Figure 25: Event log of running example, for case P1-Ta-SO1

5.2. Extracting Log Extraction Specifications

To extract a LES we need to create an algorithm or query that takes as input a selection of columns from tables, and outputs a LES as we defined in the previous section. We have to create such a query manually, for each use case, based on our domain knowledge. We explain the reasoning that goes into the construction of such a query and explain the LES query along the running example of Figure 22 and data in Figure 21.
Step 1: Get timeframe for object 1 (pallet) by joining first and last event in use case

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-1</th>
<th>De-stack Task ID</th>
<th>End-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12:24</td>
<td>DST1</td>
<td>12:47</td>
</tr>
<tr>
<td>P2</td>
<td>13:15</td>
<td>DST2</td>
<td>13:35</td>
</tr>
</tbody>
</table>

Step 2: Get timeframe for object 2 (trays) by joining first and last event in use case

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-2</th>
<th>De-stack Task ID</th>
<th>End-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ta</td>
<td>12:42</td>
<td>DST1</td>
<td>13:05</td>
</tr>
<tr>
<td>Tb</td>
<td>12:42</td>
<td>DST1</td>
<td>13:06</td>
</tr>
<tr>
<td>Tb</td>
<td>13:17</td>
<td>DST2</td>
<td>13:40</td>
</tr>
<tr>
<td>Ta</td>
<td>13:20</td>
<td>DST1</td>
<td>13:35</td>
</tr>
</tbody>
</table>

Step 3: Get timeframe for object 3 (stacking pallet) by joining first and last event in use case

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-3</th>
<th>Stack Task ID</th>
<th>End-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>SO1</td>
<td>13:00</td>
<td>ST1</td>
<td>13:10</td>
</tr>
<tr>
<td>SO2</td>
<td>13:30</td>
<td>ST2</td>
<td>13:45</td>
</tr>
</tbody>
</table>

Step 4: Join the timeframes of object 1 (pallet) and object 2 (trays), based on Task ID

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-1</th>
<th>End-1</th>
<th>De-stack Task ID</th>
<th>ID</th>
<th>Start-2</th>
<th>End-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Ta</td>
<td>12:42</td>
<td>13:05</td>
</tr>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Tb</td>
<td>12:42</td>
<td>13:06</td>
</tr>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Ta</td>
<td>13:20</td>
<td>13:35</td>
</tr>
</tbody>
</table>

Step 5: Join the recently obtained table with timeframes for object 1 (pallet) and 2 (tray) with the timeframe of object 3 (stacking pallet) based on Tray ID

<table>
<thead>
<tr>
<th>ID</th>
<th>Start-1</th>
<th>End-1</th>
<th>De-stack Task ID</th>
<th>ID</th>
<th>Start-2</th>
<th>End-2</th>
<th>Stack Task ID</th>
<th>ID</th>
<th>Start-3</th>
<th>End-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Ta</td>
<td>12:42</td>
<td>13:05</td>
<td>ST1</td>
<td>SO1</td>
<td>13:00</td>
<td>13:10</td>
</tr>
<tr>
<td>P1</td>
<td>12:24</td>
<td>12:47</td>
<td>DST1</td>
<td>Tb</td>
<td>12:42</td>
<td>13:06</td>
<td>ST1</td>
<td>SO1</td>
<td>13:00</td>
<td>13:10</td>
</tr>
</tbody>
</table>

Figure 26: Extracting the LES from various tables

Figure 26 illustrates the selection of object id and timestamp columns to obtain the LES in a few steps. We have abstracted from the details and focus on object 1, 2, and 3 (representing an inbound pallet, tray, and outbound pallet). Note that the IDs correspond directly with the objects shown in Figure 22, and the coloring again highlights to which Stacking Order the events belong. More specifically, the coloring indicates on which outbound pallet the products on each of the other objects end up. Next, we elaborate the steps taken to obtain this LES for the In-to-Out use case.

The first step is selecting the start and end events for each object from the respective columns. This is illustrated as small tables called “Object # start events” and “Object # end events”. We then join the start and end messages per object to obtain the case time-frame for each object. For the pallets this is trivial, as they have a finite lifecycle (Start: Receiving, End: Empty Pallet Exit). However, for trays this is slightly more invested, as we have to account for the infinite lifecycle. We can do the join for start/end events for trays as follows (Start/End: In Storage or Empty Tray Storage):

- We do an outer join on the Tray ID
- Sort on ascending start time, then end time
- Remove all end events that have a timestamp smaller than the start event timestamp
- Remove the remaining duplicates based on the object ID and Start time

We can do this because the first end event after the start event in consideration will always be the corresponding one for this lifecycle. We illustrate that this is indeed the case by looking at Ta. Applying an outer join (and sorting on start time, then end time) to the start and end events related to Ta we get the table on the left in Figure 27.
Following the logic, if we now remove the events/rows that have an end time that is before the start time we remove the third row (in italic, also indicated by arrows), which yields the middle table. If we now remove duplicates from top to bottom based on the first two columns, i.e. ID and Start, we remove the second row. This leaves only the correct start and end times for the tray lifecycle, illustrated in the rightmost table. This last removal holds because we sorted and hence a duplicate record means two (or more) intervals overlapping, the larger ones being the result of joining two smaller intervals, and are hence wrong.

Once we have the case time-frame for each object we are interested in we have a number of tables that is equal to the number of objects, in this case 3. We then join these tables one by one, as illustrated by step 7 and 8. Hence, we first join the inbound pallet and tray – based on the de-stacking Task ID, which we obtain from the start event. Afterwards, we join the combination of pallet and tray to the outbound pallet – based on the stacking Task ID, which we obtain based on the start event of the pallet and the Task Assign event, which includes the tray ID(s) to be used. There can be multiple instances of a tray at stacking, while it only has one instance at de-stacking, like we see for Ta. We can preserve this in our LES by using an outer join, which keeps both occurrences, but we have to make sure the occurrence of the tray at stacking is actually connected to the correct occurrence at de-stacking. If we do this incorrectly we will have overlapping intervals like in Figure 27. We can define case timeframes for objects with persistent IDs as follows:

- By using additional attributes in the data if these are available – for instance, an attribute that tells us which product is in the tray.
- If such information is not available we rely on the following logic: after a tray is filled at de-stacking, the first occurrence of this tray at stacking is the one corresponding to the de-stacking task. Thus, we can use an outer join, sort on time, and then remove duplicates, which comes down to keeping only the first match just as we described for obtaining the case time-frames for Ta in Figure 27.

The resulting table is the output of our algorithm, which is the log extraction specification as specified in Section 5.1.4 for the running example in Figure 22.

We will now formulate this approach on a more abstract level, by conceptualizing it into a few steps (on the next page). A query or algorithm corresponding to these steps must be handcrafted for the system, as the inputs are crucial and have to be selected or specified manually. The input parameters should be obtainable by following the approach as described in the previous sections. Specifically, an understanding of analysis use cases, the data source, and lifecycle models like we defined in Chapter 4 is required. On a high level the steps are quite trivial, as the details make the joining operations more or less difficult. The output is a log extraction specification as we defined in Figure 23, but then in a table format – shown again in Figure 24. Using this approach we crafted multiple queries to extract various use cases (sub-processes) from the system in the case study. To extract a specific use case, the user has to look at the objects and lifecycle definitions for that use case, and tailor the query so that it only extracts the relevant use cases and their case time-frames.
**Input:**
- Start event type and End event type (per object type)
- Timestamp columns
- Case identifier columns (Object ID columns)
- Task ID columns (or likewise)
- Any other columns that can help with linking together events
- Joining logic (i.e. first occurrence after start is end)

**Steps:**
1) Per object do the following
   a. Select start events
   b. Select end events
   c. Join the start and end events on Object ID (potentially using additional operations to define case timeframes for objects with persistent IDs)
2) Join the Object case time-frame tables one by one

**Output:**
✓ A Log Extraction Specification in the format as specified in Figure 23

With this we have answered “How to extract the specified lifecycles of objects from the data?” and “How do we make the connections between various Tasks and objects as we modeled, and extract this from the data?”, thereby answering all three questions we raised at the end of Chapter 4, and completing the second step of our ETL approach as proposed in the introduction of this chapter. Thus, we can now move towards automatically extracting event logs by using this LES extraction method, which is the third and final step of the ETL approach.

### 5.3. Extracting Event Logs

In this section we explain how we automatically extract event logs using the LES extraction. Given proper construction and extraction of the log extraction specification the algorithm for log extraction is quite straightforward. We can use the IDs and timestamps in the LES as follows:
- The (Object) ID columns can be used to construct the case identifier (and to find events during querying)
- The timestamp columns can be used to limit the query range, i.e. to extract only events from inside the object’s timeframe

We discuss the algorithm based on the ‘default view’, which is the finest granularity case identifier (on tray level), and shortly discuss the example extracted log in Figure 25 using this algorithm on the running example in Section 5.3.1. Afterwards we discuss the ‘other views’, i.e. other granularities for the case identifier, we can create by using a subset of the object ids in Section 5.3.2.

#### 5.3.1. Event Log Extraction with Default View

Below, we present the event log extraction algorithm with its input and output, after which we discuss what it does (or is supposed to do) exactly.

**Input:** Log Extraction Specification (Table)

**Algorithm** extractEventLog(LES):
1   eventLog = new List()
2   for each row in LES:
3       CaseId = concatenate(ObjectId1...n)
4       for each ObjectId column:
5           event_records = Query for ObjectId between StartTime and EndTime
6           for each Event in event_records:
7               Create Activity from some Event.Attribute
8               eventLog.append(Activity, Event.timestamp, CaseId)
9   export eventLog.csv

**Output:** Event Log
The input is the log extraction specification (in the desired format – in this case a table) we extracted in the previous section. The first step is to create a new empty list, or any data structure that can hold events in a total order. Then, for each row of the LES table, we first create a case identifier by concatenating all the Object IDs in that row. Then for each object column in that row we first query for all events between the Start and End time from the LES, that somehow include the Object ID. We query from specific tables based on the use case chosen and the corresponding lifecycle model. Then, for each retrieved event in the result set of the query, we create an activity (or something that can act as an activity) from an event attribute, in our case this could be the Event Type (i.e. the name of the table we extracted this event from). Finally, we append the event log by our newly found event, and repeat the loops until we are done. At the end we export the event log, for instance to a CSV file.

In terms of performance the algorithm is relatively slow, as there are three nested loops, resulting in a runtime of roughly $O(n^3)$, with $n$ as the number of retrieved events. If aiming for optimization we could divide the LES table into parts and run the event log extraction algorithm in multiple parallel instances, after which we only have to append the exported event logs to each other (and sort). However, for our research purposes, this simple algorithm suffices. Moreover, the database for the system in the case study (Splunk) queries using a pipeline-like iterative language. So, instead of having three loops it is more like running three algorithms after each other, where they are each other’s input. Thus resulting in better performance at roughly $O(n^3 \times m \times k)$ ($\#rows \times \#objects \times \#events$), at the cost of losing the opportunity to easily distribute and parallelize the work.

Figure 25 illustrates the resulting (excerpt of an) event log in CSV format in a table, when applying this algorithm on the running example of Figure 22 and thus the LES at the end of Figure 26. It specifically shows the case consisting of P1, Ta, and SO1. We have added a column that includes some information on the location or destination of an object, which is commonly available as an extra attribute in the data we have – as we illustrated in Figure 13. As can be seen, this shortened trace of one case is already quite long. Moreover, a lot of the events are solely about the Tasks, and not so much about the physical flow of the system. Next, we go over the various options for the case identifier, i.e. the other views on the process we can create.

### 5.3.2. Event Log Extraction with Other Views

In Figure 25 the case identifier is ”P1-Ta-SO1”, but as we shortly mentioned in Section 5.1.2 there are more options for the case identifier. Considering the 1:n and n:1 relations between objects, we identified the case IDs that can be chosen. Each of these case IDs brings its own view on the process, which we elaborate below. For the complete ‘In-to-Out’ use case we list at all the possibilities:

- **1:1:1**: Products from 1 pallet are put into 1 tray, which is used for 1 stacking order. This represents the finest granularity, and the most probable case ID for root-cause analysis.
- **1:m:1**: Products from 1 pallet are put into multiple trays, which are then all used to fulfill 1 stacking order. This is not very likely, but when an order consists solely of 1 type of product this is possible, or when making a very specific selection – which would require additional processing of the event log.
- **1:m:m**: Products from 1 pallet are put into multiple trays, from which multiple different stacking orders are fulfilled.
- **n:mm**: Products from $n$ pallets are put into $m$ trays, which are used for 1 stacking order. This is simply the buildup for 1 stacking order, and as such can pose interesting to find issues related to a specific order.
- **1:1:n**: Products in 1 tray are used in multiple stacking orders. This is not very likely but can occur (like with Ta in the running example).
- **n:mmk**: Products from $n$ pallets are put into $m$ trays, which are used for $k$ stacking orders. This cannot be used as this means connecting an arbitrary number of pallets to an arbitrary number of trays, which are not necessarily actually connected, and then connecting this to an arbitrary number of stacking orders. This will lead to an overly complicated model with meaningless relations (edges). If taking all pallets and trays this also cannot be used as then there is only 1 case in the entire log, which also results in a large and overly complicated model. A better option is to have multiple 1:1:1 cases in 1 event log and look at the aggregated model, which allows to see the general flow of the system and processes.
- **n:1:m**: Products from $n$ pallets are put into 1 tray, which is then used for $m$ stacking orders. This is impossible as each tray can only carry 1 product type and is only filled once in a lifecycle.
- **n:1:1**: Products from $n$ pallets are put into 1 tray, which is then used for 1 stacking order. Again this is impossible, just like n:1:m. However, both n:1:m and n:1:1 could be possible if the definition of the tray’s lifecycle changes.
Note that we can adapt line 4 of the pseudocode algorithm for log extraction in Section 5.3.1 to build *multiple case IDs*. We then have *multiple case ids* in the event log, from which the user can *choose* one when loading the event log into a tool. As an example, we show the five different *case id types* for the running example. In general, whenever an *n* or *m* occurs, we leave out the corresponding ID in the concatenation. Considering the LES in Figure 26, if we put the resulting case IDs in a table per *case id type* as above, we get the following case IDs. Thereby, from left to right in the table, there would be 4, 3, 2, 2, 3 unique cases.

### Table 5.3: Overview of different case identifiers for the running example

<table>
<thead>
<tr>
<th>Case ID 1:1:1</th>
<th>Case ID 1:n:1</th>
<th>Case ID 1:m:n</th>
<th>Case ID n:m:1</th>
<th>Case ID 1:1:n</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1-Ta-SO1</td>
<td>P1-SO1</td>
<td>P1</td>
<td>SO1</td>
<td>P1-Ta</td>
</tr>
<tr>
<td>P1-Tb-SO1</td>
<td>P1-SO1</td>
<td>P1</td>
<td>SO1</td>
<td>P1-Tb</td>
</tr>
<tr>
<td>P2-Ta-SO2</td>
<td>P2-SO2</td>
<td>P2</td>
<td>SO2</td>
<td>P2-Tb</td>
</tr>
<tr>
<td>P1-Ta-SO2</td>
<td>P1-SO2</td>
<td>P1</td>
<td>SO2</td>
<td>P1-Ta</td>
</tr>
</tbody>
</table>

Hence, we can choose between 5 different possible case ID types when extracting the log. Each of these case IDs brings its own view at the process. As we know from earlier work, extracting events through a 1:n relation leads to convergence and divergence.[11] We explain how we try to handle this later in Chapter 6. Which case ID to use when depends on the analysis questions and goal – and thus depends on which use case we extract. Recall that we have four use cases: Inbound, De-Stacking, Stacking, and In-to-Out. However, none of these use cases is necessarily focused on the 1:n relations. Moreover, for each use case that includes multiple objects we can opt to only look at one of the objects. In the case of warehousing one might want to pinpoint what caused the delay of a shipment (i.e. the delay of a stacking order). For this particular question the *n:*m:1 case id could be used, after which a filter to get the specific outbound pallet ID could be set up. We examine the different case IDs a bit more elaborately in Chapter 6 and 7.

### 5.4. Extracted Event Logs

In the previous section we completed the last step of our ETL approach from the introduction of this chapter. Using the LES we were able to create a relatively generic algorithm/query that allows for automatic extraction of event logs given a LES as input, thereby answering RQ3.

We used the event log extraction approach to devise queries for the case study system to obtain a number of different event logs. We elaborate shortly on the two main event logs we extracted, and show some statistics for these logs.

We have two (initial) logs that contain data from the 16th of May 2018 and cover the entire process, thus containing the In-to-Out use case. For the case identifier we used the 1:1:1 type, to have the lowest level case identifier. The table below holds some statistics on the two event logs.

**Table 5.4: Statistics on 2 event logs**

<table>
<thead>
<tr>
<th>Log</th>
<th>#Events</th>
<th>#Cases</th>
<th>#Variants</th>
<th>#Activities</th>
<th>Events/case</th>
<th>Mean</th>
<th>Median</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31265</td>
<td>258</td>
<td>258</td>
<td>486</td>
<td>98-177</td>
<td>5.9 hrs</td>
<td>5.9 hrs</td>
<td>4 hrs 29 m</td>
<td>8 hrs 32 m</td>
</tr>
<tr>
<td>2</td>
<td>9407</td>
<td>258</td>
<td>258</td>
<td>448</td>
<td>31-57</td>
<td>5.9 hrs</td>
<td>5.9 hrs</td>
<td>4 hrs 29 m</td>
<td>8 hrs 32 m</td>
</tr>
</tbody>
</table>

Event log 1 is a ‘full’ event log, i.e. including all event types, whereas event log 2 only holds activities that relate to the physical flow of objects. We will elaborate on this physical flow log in Chapter 6. LES extraction took approximately 30 seconds, where most of the time is spent on parsing the query. Event log extraction took approximately 20 minutes for the full log, and around 10 minutes for the physical flow log. We will further investigate the data in these two event logs in Section 6.2 where we do data exploration. Afterwards, we propose improvement methods, elaborate a bit more on the chosen tooling for analysis, and evaluate based on quality criteria.
6. EVENT LOG PRE-PROCESSING

In the previous chapter we described one way to extract event logs from material handling systems with automated batching, from a specific warehouse automation system (WAS) as described in the preceding chapters, thereby answering the second research question. Note that here we cross the boundary between theoretical study and case study, as the results we obtain are from the Vanderlande system in the case study. As such, the discussion in the following 2 chapters is focused on the implications for this warehouse automation system specifically. However, as the chosen warehouse automation system is a real-life example of the concepts we have been building up in the previous chapters, the discussion should be relevant for the entire quadrant of material handling systems with multiple case identifiers. In this chapter, we can now turn to our third research question, to attain insights in the applicability of process mining as follows:

(III) Given a set of event logs that contain data of a MHS we would like to explore the possibility of applying and adopting current process mining solutions to attain insights in the applicability and limitations of these solutions.

Thus, the goal we want to achieve is that engineers can obtain insights into the behavior of WASs using existing process mining tooling. We identified four main questions to achieve this goal:

1. How does existing standard process mining tooling behave and where does it fail?
2. Can we pre-process the data to avoid the identified failing?
3. Are there tools that avoid the identified failing?
4. As we have multiple views on the data (i.e. multiple event logs), can all views on the system be analyzed with the same pre-processing and tooling or do we need a more specific analysis methodology of specific pre-processing and specific tooling for the different views?

To answer the first two questions we first devise a list of quality criteria, which we discuss in Section 6.1. Then, in Section 6.2, we explore the event logs in Disco, to identify where quality criteria are not met, i.e. to identify limitations. From here we propose various data pre-processing approaches to improve quality, which we systematically describe in Section 6.3, based on the problems they intend to solve. We answer the remaining questions in Chapter 7, where we first identify tools that may avoid the identified failings, after which we set up and execute a combinatorial study to evaluate the tools and pre-processing approaches.

6.1. Quality Criteria

To validate the usefulness and value of each of the resulting models we devise a list of criteria that must be met. The criteria range from more model oriented to tool oriented, in the sense that some criteria are not so much about the quality of the model but more about the functionality of the tool or the capability of the tool. Moreover, the criteria do not always all have to be met as depending on the analysis question some criteria might not come into play at all. Hence, these criteria form a baseline of what we deem important qualities for the models, but are nevertheless not complete or restricting and can very well be extended or adapted to better suit a different analysis purpose or system. The criteria are split up in two main categories as follows:

1) General legibility and usefulness of the model
   QC1) The model shall not be a so-called spaghetti model
   QC2) The model shall be readable and understandable for an engineer
   QC3) Any edge in the model shall only exist if it relates to the physical flow of an object
   QC4) The model/tool shall be able to handle a large number variants

2) Analysis value of the model/tool
   The model/tool shall be able to:
   QC5) show bottlenecks in the process
   QC6) show deviating cases (outliers)
   QC7) portray/handle concurrency (of the multiple objects)
   QC8) distinguish between the different objects
   QC9) show the handover from pallet to tray (if both are present)
6.2. Data Exploration

Having extracted event logs and established quality criteria, we now need to investigate whether existing tooling produces results that meet these quality criteria. For this, we designed the following explorative study. We extracted the In-to-Out use case (the entire process) for 1:1:1 case identifiers, as we described in Section 5.4. We extracted this specific log as it gives an overview of the entire system and process, hence including handover between pallet and tray (batching) at de-stacking and stacking, multiple different objects and thereby multiple possible case ids, and as such covers all relevant aspects we describe in the research problem, except for convergence, which we come back to later. The obtained log has 258 trays, 7 inbound pallets, 19 outbound pallets, 486 activities, 258 cases, and 31265 events. More statistics are in Table 5.

In Section 5.4. We examine this log in Disco, after which we already apply a few necessary pre-processing steps, which we will explain along the way. Note that we do not go into details but purely look at things like model size and overall readability.

Figure 28 shows the model obtained from this In-to-Out event log using Disco in the standard setting of 100% activities and 0% paths (only the most frequent paths).

We can immediately disqualify this model based on QC1 and QC2, as this model is clearly a so-called spaghetti model and is difficult to read. Not only are there many edges, there are also so many activities that the entire tool slows down. Moreover, given the illegibility of the model, more quality criteria are likely also violated, like QC5 or QC9. The many activities we see are mostly events of a tray being stored in different physical locations. However, we are only interested in the fact that the tray is being stored, not so much where exactly; at least for the process model. There are more places where this happens, like entry and exit paths towards modules. Hence, the first step in achieving better (smaller) models becomes aggregation on the physical level. For instance, we could aggregate away the many storage locations that are shown in the model and replace them with one activity “Storage”. We will apply at least this much aggregation (renaming) on every event log as otherwise regardless of what approach is taken the model will simply be too large to possibly analyze.

Thus, aggregation becomes a necessary first step in analyzing the event logs. We will explain how we achieved this aggregation later in Section 6.3. The resulting model from the aggregated full event log is shown in Figure 29. The number of activities is reduced from 486 to 103 which is a significant decrease by almost 80%. The model is more legible than the one in Figure 28. However, there are still a plethora of issues, the first being that this model is still at “0% paths” and the 100% paths model still violates QC1 and QC2, and we could argue that the 0% paths model validates (at least) QC1.

Furthermore, the activities highlighted with a red circle have a high degree of ingoing and outgoing edges, i.e. there are a lot of ingoing and outgoing edges. This makes it quite hard to follow the actual flow of the process, as it is not clear which path to take. Moreover, these edges go towards many different activities in different places in the process. Upon further inspection it seems these activities are (often) the Task messages. It makes sense that these Task events have such a high degree as the tasks can be assigned at many points in the process – consider again Figure 19 in Chapter 4 which has e.g. multiple Transport tasks at various places in the process. More specifically, the task messages suffer divergence issues, i.e. the activities are related to multiple different cases, with differing context. For example, the activity labeled with A refers to a transport task to move the pallet to and from storage, but also refers to a transport task to move a tray to and from storage, or to and from de-stacking or stacking, which are all called “Transport Task”. Hence, this one activity refers to many different contexts of that activity.
The divergence obscures the analyst from following and analyzing the process. To focus on the physical flow in the system we can filter out the Task related events. Doing so leaves us with only the Location Status and the Module Status update events as we described in Chapters 3 and 4, which further reduces the number of activities from 103 to 63 (see Table 5.4). Removing the Task events can be done either during log extraction or afterwards using filtering (e.g., in Disco). Regardless of the chosen approach, the resulting models should be identical as the remaining events in the traces should stay the same. Figure 30 shows two models for the physical flow event log. The 0% model is clearly legible and does not violate QC1 and QC2 like the earlier discovered models. However, the 100% model still has a high number of edges, thereby violating QC1 and QC2. This can be explained by the concurrency of a pallet and a tray as we explained in Figure 18 in Section 4.5. The 100% model of Disco draws a directly follows graph that renders all the possible temporal interleavings of tray and pallet events, leading to the strongly connected subgraph, rather than rendering only the concurrent physical flow of trays and pallets. Thus, the model also violates (at least) QC3 and QC7. Hence, to comply to the quality criteria, in Disco we could choose to simply use the 0% paths model. However, this results in filtering out a lot of paths, more specifically filtering out the less occurring paths—the outliers. As we are usually interested in outliers, this is not a proper solution as it leaves them out and thereby violates QC6 as well. Thus, we have to find different solutions for handling concurrency (parallelism), divergence, and convergence in the models.

So far we have identified a few major challenges for the model analysis, which are reflected by the quality criteria:

- The number of activities in the model (QC1, QC2, QC4, QC6)
- The number of edges in the model (QC1, QC2, QC4, QC6)
- Due to concurrency, divergence, and convergence: (all: QC1, QC2)
  - The number of meaningless or false edges in the model (all) (QC3, QC6, QC7)
  - Render/discover the concurrency between objects (concurrency) (QC7, QC8, QC9)
  - Task messages reoccurring throughout the process (divergence) (QC3)
- Obtaining correct performance information (QC5)

To reduce the impact of some of these challenges we already introduced two approaches for improvement. These are aggregation and ‘removal of task messages’, or more generally: selection, through event log pre-processing. We further elaborate the pre-processing approaches and introduce new approaches in the following section.

Figure 29: Model of the full process, with an aggregated event log

Figure 30: Models for the physical flow event log
6.3. Improvement Approaches

During the data exploration in Section 6.1 we observed that event log pre-processing helps to significantly improve model quality. In this section, we systematically discuss a number of log pre-processing approaches and their effect on the event data. Standard operations to modify data are: aggregation, selection, projection, renaming or relabeling, and extraction of different logs (i.e., for different use cases). For each of these operations we discuss a few specific approaches to improve the model quality. We will not show the results of each of the approaches; we discuss those in Chapter 7. In Table 6.1 we indicate which operation (column) was successful in addressing one or more problems (row), and in which subsection it is discussed. Each respective section elaborates on the approach and how we implemented it.

Table 6.1: Pre-processing approaches and the problems they can solve, and in which section they are discussed

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approach</th>
<th>Aggregation</th>
<th>Selection/Projection</th>
<th>Renaming</th>
<th>Different data source</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Activities</td>
<td></td>
<td>6.3.1</td>
<td>6.3.3</td>
<td>6.3.2</td>
<td>6.3.4</td>
</tr>
<tr>
<td>#Edges</td>
<td></td>
<td>6.3.1</td>
<td>6.3.3</td>
<td>6.3.2</td>
<td>6.3.4</td>
</tr>
<tr>
<td>#Meaningless/false edges</td>
<td></td>
<td>6.3.1</td>
<td>6.3.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrency</td>
<td></td>
<td>6.3.3</td>
<td></td>
<td>6.3.4</td>
<td></td>
</tr>
<tr>
<td>Divergence</td>
<td></td>
<td>6.3.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convergence</td>
<td></td>
<td>6.3.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correct performance info.</td>
<td></td>
<td>6.3.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.3.1. Aggregation

The main use of aggregation is reducing model size. Aggregation can also get rid of meaningless or irrelevant edges by putting these inside one activity. We discuss two approaches for aggregation: unifying activity names and grouping events.

Unifying Activity Names

To significantly reduce model size we can aggregate through unifying activity names. Figure 31 illustrates unifying activity names through an example of storage locations. Another applied way to do so is unifying activity names...
based on which module sent the message and grouping modules of the same type into one. We can aggregate during log extraction or afterwards using a separate pre-processing script, both of which we shortly elaborate below.

![Figure 31: Renaming various storage locations to one general 'stored' activity to reduce model size](image)

During log extraction, when we create an activity in the algorithm, we can unify activities by for instance using a simple if-clause that checks for ‘Storage’ being in the event data, if so we name the activity “Stored”, like in Figure 31. Depending on the database tool and the data itself more elaborate aggregation could be done, but it might be easier to do so in a pre-processing script as follows. Using a separate pre-processing script allows us to rapidly try out various approaches on multiple event logs, which speeds up the pre-processing significantly over e.g. using Excel to manually change activity names. A function/algorithm for unifying activity names can be very trivial: pass over the event log once and check each event for a certain condition, if the condition is true: change the activity name, otherwise do nothing and continue. Hence, a ‘unifying function’ like this should be relatively easy to recreate in any programming language.

High-Level Aggregation (Grouping Events)
Another way of significantly reducing the model size is by applying aggregation (grouping events) to obtain a more high level version of the model. For instance, we could aggregate the activities into the respective modules each activity belongs to, to get an overview of how the modules interact on a high level. We can define multiple levels of aggregation to go from a very high level model all the way back to the original model. Each of these levels could bring its own advantages or disadvantage. Moreover, aggregation like this can also remove irrelevant edges as they then reside inside one activity.

6.3.2 Renaming/Relabeling
As proposed by Xixi Lu et al. [18] adding context to activities can (partially) solve data divergence. Recall that data divergence occurs when one case is related to multiple events of the same type. In the data of the warehousing system this shows up in the Tasks. For instance, throughout the lifecycle of a tray or pallet they are assigned multiple transport tasks. The transport tasks are different as they have a different destination, but in the event log they are represented by the same activity. Thus, if we add context to the (label of) the activity or event we can explicitly distinguish between the different tasks, which should solve the data divergence. We devised two ways of adding context to an event log, which are adding numbering and adding explicit information.

We can add a numerical context (a count) by counting the activities in a trace and adding the respective count number to its label. This removes unwanted loops at the Task events, but also removes all loops from the process. Adding explicit information from the event to the label, so that it can be distinguished, like adding the destination of a Transport task to the label of the event, changes the context for each Transport task. However, when we add context in we do not necessarily improve the model. Considering that Tasks can be assigned largely independent from the physical flow this means that the chance of having two tasks with the same number having the same actual context is small for a large number of traces, let alone all tasks with the same number. Moreover, it can still occur that a Transport task towards a specific destination is assigned at a different moment in the lifecycle of an object. Hence, both approaches for adding context have significant drawbacks and might not work as well as intended.

6.3.3 Selection and Projection
Through selection and projection we can filter in or out specific activities/events that we deem interesting for our analysis goal. We can select and project based on the objects (tray, pallet), activities (i.e. task events), and other attributes in the events. We propose five different selection/projection approaches to improve the model quality, listing the problem each of the approaches solves; note that all also reduce the model size.

Physical Flow vs. Full Log
To solve the problem of ‘meaningless/false edges’ and partially solve divergence, we can choose to project a log that does not include Task events, i.e. we select a subset of activities. We can do so during log extraction, but this might require extraction of a new log, which might be time consuming. An easier approach is to load the complete event
Process Mining for Systems with Automated Batching

log into a tool like Disco, and then create a filter that only includes the desired activities. This *physical flow log* is used often – and is the preference of the engineers – as it greatly reduces the size of the model, focuses on what happens to the *objects* in the system, and increases the ease by which the model can be analyzed as we have already described in Section 6.2.

**Separating Objects**
We can solve *convergence, divergence, and concurrency* issues by projecting an event log that only contains *one object type*. Doing so ‘solves’ the main challenges of process mining for warehouse automation system as it reduces the case identifier to a single case identifier. However, by reducing the event log to only contain 1 object we also limit the analysis potential. For instance, we can no longer study the relation between a pallet and a tray and we can no longer study the impact the physical flows of these objects have on each other.

**Specific Selection**
To reduce the size of the model and mostly to solve concurrency between objects we can do very specific projection, i.e. we keep only a specific subset of events from one object and keep all events from the other object. For instance, we could keep only the pallet events up until and including the handover point between pallet and tray, and remove all the other pallet events. This leads to a reduction of the parallel flows in the process, hence removing a lot of unwanted edges, but still allows us to analyze the relation between pallet and tray at de-stacking. This kind of specific selection is very powerful for improving model quality, but requires high effort and domain knowledge.

**Task Model**
In contrary to the physical flow model, we can also project an event log that *only* contains the task messages. Doing so can remove *meaningless edges* (and reduce the size of the model), but this does create a very specific focus. This model could help with analyzing the behavior of the Conductor module, and could show any irregularities in the (high level) process that is being executed.

**Splitting Log**
Finally, to obtain *correct performance information* in for instance Disco, it is important that we have a way to handle the storage events. If we do not address this the storage events will always get identified as the “bottleneck”, whereas this is an *expected* waiting time and thus should not be identified as such. One way of addressing this is by splitting the logs into two at the storage event and thereby creating two new logs. We can then separately analyze the resulting models, which allow tools like Disco to (more) correctly highlight the bottlenecks.

**6.3.4. Different Data Source**
The last approach for *reducing model size and complexity* is to select a different use case to extract, depending on the analysis question/goal. This gives us a *new and possibly smaller event log* to work with. As we described in the previous chapters each use case has its own start and end times for the lifecycle of an object. Thus, we can adjust the LES extraction in Section 5.2 to only return the required information for a specific use case. Then, the log extraction query can be kept exactly the same as the inputs are the same, only the data retrieval window is reduced by the input parameters. Moreover, projecting or filtering a use case from the entire log is also possible but might be more difficult than extracting it separately.

**6.4. Conclusion**
In this chapter we identified the main challenges for process model quality through data exploration in Disco. To solve these challenges we proposed various pre-processing options, which we implemented in a pre-processing script. The next step is to evaluate the pre-processing approaches and other tools:

1) How does this pre-processing impact the model quality in a standard tool (Disco)?
2) Do other tools offer better/more insights compared to a standard tool to overcome specific challenges or address specific use cases?

By addressing these two questions we can provide an answer to the remaining two questions we identified in the introduction of this chapter, and thereby answer the third research question.
7. ANALYSIS AND EVALUATION

In this chapter we aim to achieve a selection of tooling and pre-processing that allows to get insights for relevant use cases on the warehouse automation system data, that can overcome the challenges identified at the end of Section 6.2, thereby complying to the quality criteria identified in Section 6.1 and answering our third research question. We recall the identified challenges below:

- The number of activities in the model
- The number of edges in the model
- Due to concurrency, divergence, and convergence:
  - The number of meaningless or false edges in the model (all)
  - Render/discover the concurrency between objects (concurrency)
  - Task messages reoccurring throughout the process (divergence)
- Obtaining correct performance information

To evaluate which combination of view on the data (use case), pre-processing, and tooling gives the best resulting model with respect to the quality criteria in Section 6.1, we conducted a combinatorial study. We first describe the tools we considered for the analysis in Section 7.1. Then, in Section 7.2, we elaborate on the setup of the combinatorial study. Finally, in Section 7.3 we provide the results of the study, after which we conclude in Section 7.4 where we summarize the findings for each tool and each pre-processing approach and address some overall conclusions.

7.1. Tooling

The tools that we will consider during the analysis are a standard industrial process mining tool (Disco [6]), results of previous work, i.e. the Performance Spectrum Miner [13], or are deemed relevant research tools in the ProM framework [7] [8]: Inductive Visual Miner [4], Interactive Data-aware Heuristics Miner [5], and the Log to Model Explorer [18]. Each of these tools has its own advantages and disadvantages, which we shortly elaborate.

Disco
Disco is an industrial process mining tool developed by Fluxicon [6] as we introduced in Chapter 2. The main advantage of using Disco is the ease of use and the speed at which an initial process model can be discovered. Moreover, as this is the tool Vanderlande uses, it is inherently necessary to evaluate the approach we present in this study on this tooling. The main disadvantage of using Disco is that it creates a directly-follows-graph and as such is not capable of handling parallelism. Another slight disadvantage is that it does not allow the user to rearrange the activities in the model.

ProM: Inductive Visual Miner and the Log to Model Explorer
The Log to Model Explorer (L2MExp) allows the user to interactively explore and mine event logs using clustering, relabeling, and filtering. The possibility of discovering multiple variants of the same model, due to clustering, can pose useful. Moreover, the ability to automatically relabel events based on the context could help solving data divergence. The discovery algorithm in the L2MExp is the Inductive Visual Miner (IVM), and as such we evaluate them together. The IVM is capable of handling parallelism and discovers a model that balances precision and fitness. A disadvantage of using the IVM is the visualization, which often creates a long horizontal model for large traces, which makes it hard to capture the model on 1 screen. A downside of using the L2MExp is that it is not very straightforward and understanding the behavior of the tool might be too in depth for engineers at Vanderlande.

ProM: Interactive Data-aware Heuristics Miner
We mainly use the Interactive Data-aware Heuristics Miner for its capability of discovering C-nets, or causality nets, which allow the user to find the causality between certain (groups of) activities. This leads to very good handling of parallelism. A downside of this tool is that it is not very easy to use and the entry threshold is quite high because it offers many discovery settings.

Performance Spectrum Miner
The Performance Spectrum Miner was specifically built to handle data from material handling systems, albeit with single case identifiers. Nonetheless, this makes it an excellent candidate for analysis. Moreover, proving its usefulness can also help adoption of the tool within Vanderlande. A downside of using the tool right now is that it is still in the research and development phase and as such the GUI is very basic. However, the tool itself works properly and all desired functionality is there.
7.2. Combinatorial (Case) Study Approach

In the combinatorial study we will explore a set of approaches and tools on a specific set of event logs as shown in Table 6.1. We explain the “approach” numbers in Table 6.2, which shows the general approach and in which section the specific approach was elaborated. We annotate the use case in ( ) and the case id type in [ ]. Note that all event logs use a 1:1 case identifier (on the tray level) unless specified otherwise, and that if there is no use-case annotation in the event log column the event log includes events for the In-to-Out use case.

All but the bottom two event logs are based on either the ‘full flow’ log or the ‘physical flow’ log as we described in Section 5.4. More specifically, the “Full Model” and “Physical Flow” logs are self-explanatory and the remaining logs are all based on the physical flow log. We mainly considered the physical flow as it was identified as preferable over the full flow by the Vanderlande engineers, due to the divergence issues the Tasks bring, and is in general easier to analyze due to its smaller size, without losing critical information (unless we specifically want to analyze the Tasks, which is covered in the “Order Models”).

The bottom two event logs in the table highlighted in italics come from a different data source. The data for these logs was obtained through an emulation setup at Vanderlande where we specifically inserted a temporary failure in the system. We then extracted the event log using the queries we built for the initial event log extraction. The aim of these two analyses is to see if we can find this issue using the developed process mining techniques, which we do so using Disco and the PSM. Doing so we are able to show that the approaches developed in this thesis together with existing process mining tools can indeed lead to insights and allows for root cause analysis.

Table 6.1: Which approaches have been applied to get a log and in which tool these have been examined

<table>
<thead>
<tr>
<th>Event Log</th>
<th>Approach</th>
<th>Disco</th>
<th>iDHM</th>
<th>Ind. Miner &amp; L2MExp</th>
<th>PSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Model (in-to-out)</td>
<td>-</td>
<td>6.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregated</td>
<td>1a</td>
<td>6.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numbering Context</td>
<td>1a; 2</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific Context</td>
<td>1a; 3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Physical Flow (in-to-out)</td>
<td>4 or 5</td>
<td>6.2</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Aggregated</td>
<td>4 or 5; 1a</td>
<td>6.2</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Aggregated [n:n:1]</td>
<td>4 or 5; 1a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregated [1:n:n]</td>
<td>4 or 5; 1a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregated [1:n:1]</td>
<td>4 or 5; 1a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aggregated [1:1:n]</td>
<td>4 or 5; 1a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Order Model</td>
<td>4 or 5</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Numbering Context</td>
<td>4 or 5; 2</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific Context</td>
<td>4 or 5; 3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pallet Projection (Inbound)</td>
<td>4 or 5; 1a</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific Pal/Tray Projection (De-Stacking)</td>
<td>5; 1a; 4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-Level Aggregation 1</td>
<td>1b</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High-Level Aggregation 2</td>
<td>1b</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split at Storage (Tray)</td>
<td>5; 1a; 4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Split at Storage (Pallet)</td>
<td>5; 1a; 4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tray Projection (Stacking)</td>
<td>0; 5; 1a; 4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stacker Instance Comparison (Stacking)</td>
<td>0; 5; 4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 The source data itself remains confidential and cannot be made available.
Table 6.2: Legend with approach numbers and the sections in which these are described

<table>
<thead>
<tr>
<th>#</th>
<th>General Approach</th>
<th>Described in Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No module instance aggregation</td>
<td>6.3.1</td>
</tr>
<tr>
<td>1a</td>
<td>Unifying location names</td>
<td>6.3.1</td>
</tr>
<tr>
<td>1b</td>
<td>Process aggregation</td>
<td>6.3.1</td>
</tr>
<tr>
<td>2</td>
<td>Add context (numbering)</td>
<td>6.3.2</td>
</tr>
<tr>
<td>3</td>
<td>Add context (specific)</td>
<td>6.3.2</td>
</tr>
<tr>
<td>4</td>
<td>Projection</td>
<td>6.3.3</td>
</tr>
<tr>
<td>5</td>
<td>Smaller Log Extraction</td>
<td>6.3.3</td>
</tr>
</tbody>
</table>

We made the selection of logs as in Table 6.1 because they contain and cover all the challenges we identified in Section 6.2 as follows:

- Any of the logs addresses the size of the model in some way; some more specifically than others
- We have already seen the challenges that arise due to parallelism in the logs marked with “6.2”
- The logs with 1:n case ids will show the difficulty of handling convergence
- The ‘context’ logs will show the difficulty of handling divergence
- The ‘split at storage’ logs will show how we obtain correct and valuable performance information
- The projected inbound, de-stacking, and stacking logs show the various (smaller) use cases

Thus, this combination of logs provides a view on all the challenges we have identified throughout the thesis and includes all use cases as identified in Chapter 3 and 4. Moreover, this selection of logs allows us to evaluate all the proposed pre-processing options to improve model quality. Note that we had to do aggregation on the physical level by unifying the location names in the activities for most of the event logs, as otherwise the model was simply too big regardless of the other pre-processing done. Furthermore, as we explained in Section 6.3, in many cases it is possible to use a projection instead of extracting a smaller event log, hence the “4 or 5”.

We chose to evaluate the iDHM, and the IVM and L2MExp, on the physical flow event log mainly to try and handle the concurrency better than Disco. From there we were able to find the advantages and shortcomings of using both tools. For the PSM we used a more specific event log as described above, where we aimed to find a specific issue so that we could specifically highlight the value and potential pitfalls of doing analysis with the PSM, instead of only showing whether ‘it is possible’ to use the PSM.

We evaluated the results using the criteria as listed in Section 6.1. Some criteria might only be met partially depending on what we are looking at. More specifically, some criteria also play into the ability of an analyst or engineer to execute certain tasks with help of the model, and are thus quite subjective. If in any case a criterion is not met, but we can give reasonable arguments as to why the model still makes sense or can be of value, we will not disregard the pre-processing approach and model. Moreover, we evaluated the models for readability, understandability, and general use with various experts, specialists, and engineers at Vanderlande, after evaluating them ourselves using the previously created conceptual models and the gained understanding of the system.

7.3. Results

In the following, we discuss the results of the combinatorial case study. We report the results in the following way. We do not show and discuss the original and aggregated full flow and physical flow model, as we already showed them in Section 6.2. We will not mention whether each criterion is met or not for each of the models as that would be too extensive and repetitive discussion. Rather, we highlight the main problems and/or advantages of each of the models and pre-processing approaches and try to be as concise as possible. Most sections will not show the model itself. Instead, for each of the sections in which we present a +/- list we refer to Appendix B where the full discussions can be found. The sections on the alternate tools and 1:n relations will hold the complete discussions as they provide more interesting discussions. In the concluding Section 7.4 we will summarize the findings and the reasoning as to provide an answer to the third research question by giving insights on the applicability and limitations of the current process mining tooling on data of warehouse automation systems.
7.3.1. Full Model – Numbering Context

+ Good for analyzing trace models of 1 case at a time
+ Divergence has less impact but is not solved completely
± Very sequential and long, however expected due to the removal of loops
  – Still a large model
  – Bad for larger scale analysis (i.e. more than 1 case)
  – False/meaningless edges remain due to not handling concurrency

7.3.2. Full Model – Specific Context

+ Some nice patterns can occur, like in Figure 32
± Still a bit lackluster due to many edges surrounding the patterns
  – So-called spaghetti model (violates at least QC1 and QC2)
  – By reducing the size of the model we mostly take away the need to add context
  – False/meaningless edges

![Figure 32: Cutout of full model, showing a nice pattern](image)

7.3.3. Order Model

+ 0% paths model is very clear
+ 100% paths model is slightly less clear, but still legible, thus can help with finding deviating flows
+ Allows for analyzing high-level behavior of system (and that of the ‘Conductor’)
± Minor use case at Vanderlande
  – No specific information available, so for e.g. root cause analysis another model must be analyzed as well

7.3.4. Order Model – Numbering Context

+ Gives a clear idea of just how many tasks are sent and executed for 1 execution of in-to-out flow
+ 100% paths model also gives a clear view on how many different execution orderings there can be
  – Counting tasks can also be done using a simple query or pre-processing script

7.3.5. Order Model – Specific Context

± Allows to follow orders on a more physical level but requires exceptional domain knowledge to do so
± Allows to do a bit of load balancing analysis
  – 100% still a so-called spaghetti model
  – Both ‘advantages’ can be done better using other logs/models (e.g. actual physical flow logs)
7.3.6. Inbound Pallet Model (Projection)

Figure 33: Inbound pallet model (projection)

- Perfectly legible 100% paths model
- Allows to analyze the inbound pallet process very precisely
- Somewhat expected as there is only 1 case identifier (the pallet id)
- Specific for the ‘Inbound’ use case

7.3.7. Specific Pallet/Tray Projection (De-Stacking)

- Allows to analyze handover from inbound pallet to tray
- 100% paths still compact/legible
- Removing exit of pallet makes it impossible to detect if something goes wrong at the pallet exit, but this is determined to be very unlikely by experts, due to redundancy in the system
- Considerable effort to define the projection, which requires substantial domain knowledge

7.3.8. Tray Projection

- 100% paths model still legible (i.e. complies with QC1 and QC2)
- Allows to analyze complete tray lifecycle
- Again somewhat expected as there is only 1 case identifier (the tray id)
- Suboptimal layout due to ‘storage’ happening twice in the lifecycle of the tray, resulting in loops, but it can be reasoned by an engineer that the “loop” is after stacking – otherwise we can add context to remove this loop
- Incorrect performance information due to storage related activities happening twice
7.3.9. High Level Aggregation 1

+ Shows high level overview, like in Figure 8
+ Good to start from for e.g. beginning engineers
± Cannot dynamically change aggregation level, and thus requires multiple event logs to go from a high to a low level
− Shows false edge, highlighted in red, due to concurrency between pallet and tray

7.3.10. High Level Aggregation 2

+ Step in between high level model above and e.g. physical flow models
+ Allows to find bottlenecks/potential problems on a relatively high level, thus requiring less specific domain knowledge
− False edges due to concurrency (though little)
− Overflow part of system aggregated into 1, whereas in reality it is separate for pallets and trays, thus mixing up flows even more

7.3.11. Stacker Instance Comparison

+ Able to analyze specific issues
+ Able to obtain correct performance information
+ 100% paths model very clear
+ Animation is especially nice for business users

7.3.12. Split at Storage (Tray)

+ Able to obtain correct performance information
+ Only half the lifecycle of a pallet or tray (because we split at the storage)
− Performance analysis on <1 minute times requires overview of travel times per segment in the system

Figure 34: High level aggregation, showing the process on a high abstraction level

Figure 35: Comparison of 3 stacker module instances
7.3.13. iDHM: Physical Flow Model – Aggregated

We also suggested using the Interactive Data-aware Heuristics Miner for analysis. In Figure 36 we present an example model discovered using this plugin of ProM, based on the physical flow event log. We highlighted the start and end of the model, together with the general flow direction to make it a bit easier to follow.

The model in Figure 36 shows a perfect example of handling parallelism between the pallet and tray, as indicated by the red stars in the model. The split between pallet and tray is very nice, and joins at a logical point. The pallet exit can take some time, and so can transporting the tray to storage. Thus a join after both are done is logical, especially as the part after this join is only about the tray, and no more pallet events can occur for a case. A downside of this model is the bad layout, which is possibly due to process starting and ending with the storage (which is only 1 activity) and to fit the model onto the screen. However, as the layout is not changeable it can be a bit bothersome to analyze the process like this, especially if the model is even bigger. Moreover, the plugin might be a bit difficult to grasp as there are many options. On top of that a C-net is not necessarily known, though with general knowledge of process models this model should be fine to read. Thus, using this plugin might require training of engineers to work with ProM and the iDHM. Nonetheless, it is a good candidate for further research, as especially the parallelism between the pallet and tray can be discovered very well using this miner.

In Figure 37 we show a part of the model discovered by the Inductive Miner inside the Log to Model Explorer (L2MExp). As the Inductive Miner is the discovery algorithm used for the L2MExp we discuss them together. Figure 37 illustrates the difficulties we face with parallelism and the high variability of the traces. It splits up into 6 parallel flows, of which the bottom one splits up even more. This makes the model quite hard to read and analyze as there are so many different options to keep in mind.

The L2MExp splits the log into clusters that have similar behavior. However, due to the high variability of the traces there are more than 200 low level clusters. On the highest level there are 7 clusters, which would be doable if not for the problems the models resulting from these clusters present. For instance, in Figure 38 we see a so-called flower-loop. This is a fall-through mechanism of the Inductive Miner, which means no fitting (sub)model could be found. We cannot analyze the model if such patterns are in the model as we cannot reason about this.

Hence, we conclude that currently the L2MExp is not fit for these types of logs (high variability). When the logs have been reduced or split up the L2MExp might be of more use, but then Disco already performs a lot better as well. Thus, especially for practical use by engineers we suggest not using the L2MExp. However, for future research where the high variability could be addressed this tool might be very useful.

7.3.15. PSM: Tray Projection (Stacking)

The last tool we selected in Section 7.1 is the Performance Spectrum Miner (PSM), mainly due to its relevance for material handling systems, its availability, and its unique view on process performance. To examine the usefulness of this tool we use the data set from the emulation where we inserted a temporary failure at the exit of stacking.

After initially loading the data into the PSM we manually made an ordering of the segments displayed to get an ordering that conforms with the physical layout of the system as much as possible. Figure 39 shows an excerpt of the performance spectrum discovered by the PSM. From top to bottom the ordering is made to follow the physical layout of the system, i.e. approximately storage – retrieving - towards stacking - stacking entry - stacking exit - towards (empty tray) storage. We will analyze this performance spectrum with two more views and figures below.

Figure 37: Part of the model discovered by the inductive miner, illustrating the many parallel paths

Figure 38: Excerpt of another model discovered by the inductive miner, illustrating a flower model

Figure 39: Initial performance spectrum discovered by the PSM (after manual ordering)
Figure 40 shows the same performance spectrum as above, only now leaving out the bars that indicate the number of trays in that timeframe (w.r.t. the maximum on that segment). Thus we look only at the traces, which are now easier to see. The figure indicates what seems to be the cause of the performance issue in the black square. However, this is not the cause but the effect of the issue that is present in the two segments that are marked with a check mark. This indicates that 3 trays took a very long time to exit – right after when the failure occurred. Right now the coloring is based on the median speed of an object traveling through the segment, which is the standard setting. Using this might not always be the best solution in terms of coloring, as when simply looking for red lines one would have not found the actual cause of the issue. However, horizontal lines do indicate that something is taking quite long, but then the engineer needs to know how long such a segment is supposed to take. Hence, with some care, we can find the cause of the performance issue using the PSM.

Figure 40: Performance spectrum showing only the traces

To further highlight how analysis with the PSM can lead to interesting finds, in Figure 41 we show the same performance spectrum again, but this time we highlight an interesting pattern that seems to indicate something is going wrong. When following the process from top to bottom (1a-2-3-4-5) we see that the entrance (1a and 1b) towards stacking from the transport conveyor shows a particular pattern: some normal speed trays, followed by a few slow ones, and then after a while it repeats. This pattern is also visible in 2 (conveyor at entrance of stacking) and 3 (first conveyor in stacker). However, when looking at 4 – tray entry to tray exit (which is an accumulation of the three modules in the data) – and 5 (exit back to the transport conveyors) we find no slow trays. There are only normal speed trays with regular intervals here. Hence, it seems the stacker handles small batches at a time, where the trays are retrieved and transported faster than the stacker can handle. This is not necessarily a bad thing as having some trays waiting is okay as there is space for that in the system. However, if the number of trays that is waiting before the stacker increases, it might result in dieback of the system and potentially full lockdown of the system.

Figure 41: Performance spectrum pointing at a recurring pattern that seems to be problematic, but is not
Thus, we showed that the PSM can be very helpful for analysis on the warehouse automation system. However, there are a few drawbacks to using the PSM. First, as we explained with Figure 40, the cause of an issue may not always be where it seems to be. To resolve this other classifiers could be used, but they would have to be written manually by the engineer. Furthermore, making the sorting order for the segments is very cumbersome and requires perfect knowledge of the system layout and a translation of this layout to the information that is available in the data, which is not always the same. For instance, in our example we have location data which is based on destinations used by the system to direct objects towards certain places. However, there is an underlying network of points that is shown on the layout of the system, which have different names that should then be ‘translated’ towards the level (location) in the data before being able to specify an ordering that makes sense. If we cannot define an ordering that makes sense it may become a lot harder to spot patterns in the performance, and thereby to do analysis using the PSM in general.

7.3.16. n:n:1, 1:n:n, 1:m:1, and 1:1:n Case IDs

In the following section we present our findings on 1:n and n:1 relations using an example that uses a 1:n:n case identifier as we defined in Chapter 5. We show only 1 model and only one case identifier type because we drew the same conclusions from any other models with different case identifiers (n:n:1, 1:n:1, and 1:1:n).

In Figure 42 we show the physical flow model discovered by Disco when using the 1:n:n case identifier. To keep it legible we set the paths setting to 20%, as with 100% it is worse than the original 1:1 case identifier model in Figure 30. This is again due to the parallelism between the pallet and tray. As for the challenges the 1:n relations bring, there are many self-loops, as we extracted the data based on 1:1:1, and thus every activity is duplicated as often as the number of trays that came from 1 pallet (recall convergence). This obstructs the analysis as we can now no longer see actual self-loops as they are obscured by the artificial loops. However, from domain knowledge (and the previous models) we know that there are not many self-loops actually present in the process so this is not too much of an issue as long such information is present.

We cannot solve convergence as of now without separating the objects as we have shown in other models, i.e. in Figure 33. This is because we cannot have a trace that has pallet events, and then somehow the tray events as a group \( (n) \), and then the outbound pallet events as a group too \( (n) \). We can extract an event log in this format, but
we do not have a discovery algorithm or tool that can handle such an event log. Furthermore, we also cannot just "de-duplicate" the duplicated events, as then we would have to do so for e.g. the inbound pallet events only. However, this requires very specific input: we need to tell the algorithm exactly which events can be removed and which have to remain. On top of that we cannot automate this removal without risking removal of actual duplicate events. Hence, the only 'solution' we have right now is to split the log per object and investigate these separately. Thus a model with 1:n relations like in Figure 42 right now provides no benefits over analyzing a model with a 1:1 case id, as it only makes the analysis harder. For Vanderlande this does not necessarily pose an issue as they are mainly interested in backtracking/tracing products through the system. For that analysis goal it does not really matter that events are duplicated, as long as we can find the (cause of the) problem, which we can by using the 1:1:1 case id, as we demonstrated with Figure 35.

Moreover, the main reason we investigate the 1:n relations in these systems is because we want to analyze the interaction between the objects and the handover of products from one object to another, which we can analyze using specific selection of events on an event log that uses a 1:1:1 case identifier, like in Section 7.3.7. As there is not that much interaction in our system, discovering artifact-centric process models might not be worth the effort. More specifically, in ERP systems an object and changes to that object reside in multiple tables and so do any related objects. However, in warehouse automation systems the interaction between pallet and tray is limited to a few events at de-stacking and stacking, and thus an artifact-centric process model will likely be of less value for warehouse automation systems than it is for ERP systems. To conclude, we do not have the means to create an artifact-centric process model from our event log, which is what we need to properly portray and analyze these 1:n relations. However, analyzing the 1:n relations is not always necessary to find an answer to a question. Using the pre-processing in Chapter 6 we showed that we can get decent models and find the cause of a problem in Disco and in the PSM. Nonetheless, obtaining an artifact-centric process model is still interesting and relevant for research, and as such we present a proposal on how to do so in the concluding Chapter 9.

7.4. Conclusions

By exploring different combinations of views on the data (different event logs), different pre-processing options, and different tools, we could establish that process mining on warehouse automation systems using existing tooling can be done and can lead to valuable insights. However, doing so requires careful application of the correct view (use case, event log), pre-processing, and tool. Next, to provide an answer to our third research question, we summarize the lessons learned regarding methodology, pre-processing, and the tools. We first discuss the tools, then the pre-processing, and finally end with more general concerns.

Tools

For each of the four tools, we summarize the main advantages and disadvantages, so that it becomes clear when and how to use which tool.

Disco

Disco can be used quite well, especially when making sure there is little to no parallelism in the event log. Given some domain knowledge, the filtering options Disco provides can be used to easily do projection or selection on the event log, after which the resulting event logs can be exported easily for later use or further analysis in different tools. The performance view Disco offers can be used especially when splitting the log so that the storage does not become the automatic ‘bottleneck’. Thus, Disco has its drawbacks but given the right pre-processing steps or care it is a very adequate tool.

Log to Model Explorer and Inductive Visual Miner

Both the Log to Model Explorer (L2MExp) and the Inductive Visual Miner (IVM) are of little use for the event logs obtained in this study. Specifically due to the high variability in the traces the parallelism between pallet and tray could not be found properly by the IVM, and the L2MExp finds too many clusters, where the models discovered for the top level clusters also have issues like ‘flower-loops’. When reducing the event logs to obtain better results in these tools/plugins the results in Disco also become better, and as Disco is a simpler and more intuitive tool, it is probably better to work with Disco for practical analysis. For research purposes it might be interesting to explore the L2MExp more with event logs that hold traces that have lower variability (e.g. through aggregation).
Interactive Data-aware Heuristics Miner
The Interactive Data-aware Heuristics Miner can be used to find the parallelism between objects, but is not the most straightforward tool. As such there is a tradeoff between usefulness and practical applicability. There are many options which all change the resulting model, and not everything is clear immediately. Hence, for research purposes the tooling works well but for general analysis in an industrial setting it might be a bit too difficult to use.

Performance Spectrum Miner
The Performance Spectrum Miner works very well for the data from warehouse automation systems, and is a very promising tool to do performance analysis with. However, it is not as easy to analyze as standard process maps like in Disco, and defining an ordering for the segments can be very cumbersome – especially since for each event log it might be necessary to define a new ordering (e.g. for different use cases or subprocesses). A possible improvement could be to have the possibility to manually change the ordering in the tool during analysis (e.g. by dragging the segments) instead of defining the ordering separately, which could reduce the amount of work significantly and also allows for a bit more interaction with the tool.

Pre-processing
From the combinatorial exploration, we derived the following conclusions regarding the usefulness and applicability of the various proposed pre-processing options in an analysis of warehouse automation system (WAS) data.

Aggregation/Renaming
Aggregating or renaming certain activities to reduce the size of the model is almost always a necessity. For the data in the case study it was not possible to analyze the models without doing so. Aggregation can also be used to 'filter' out activities that are not interesting, by creating grouped activities. Furthermore, this can be taken even further by aggregating to a more high-level view of the process, which could be used to get a general idea of the process. If it was possible to dynamically change the aggregation in the event log during analysis, the aggregated models could be of more value as it would allow to expand or collapse certain parts of the process on the go.

Physical Flow vs Full Flow
We found that analyzing the ‘physical flow’ instead of the complete ‘full flow’ makes the analysis easier. Especially when interested in the physical location of objects in the system leaving out messages that do not relate to a physical location (like the Tasks) can lead to a more useful model and also decreases the size and complexity of the model considerably.

Projection/Selection/Filtering (+ Order model)
The best way to obtain better event logs and resulting models to analyze is by reducing the event logs through filtering (projection/selection). Whenever we look at only 1 object we reduce the problem down to process mining for single case identifiers, for which there exist many solutions that can be used and doing so also improves the applicability of Disco. However, in order to define a projection or selection it is necessary to know exactly which events can or cannot be left out, so the data must be known thoroughly. Luckily, projection and selection can be as easy as defining a few filters in Disco, and is thus a very useful approach for improving the process models.

Adding Context
To handle divergence in the process models we can either leave out the problematic activities (which is simply omitting the problem), or we can add context to these events so that they are no longer duplicate. We evaluated doing so by adding numbers and specific information. Adding numbering often leads to very long and sequential models, which also remove almost all loops, which can make it difficult to find actual loops. Moreover, the context of an activity with number 1 may not necessarily be the same as another activity with number 1, let alone all activities with a 1. Using specific context, like the destination of an object, makes more sense as then the context should be the same more, but does not result in very good results. Hence, if at all, adding context should be done in combination with projection or selection to obtain more specific results, without blowing up the model unnecessarily.

General Findings
Below, we list the general concerns and findings regarding process mining for WAS data, rounding up the conclusions of the evaluation.

- To do analysis on a process model of a warehouse automation system like we did in the previous section, an analyst or engineer needs to have exceptional knowledge of the system. More specifically, depending
on the type of data available, knowledge of the physical layout of the system is also necessary to reason about the discovered process models.

- To do performance analysis it is important to have some sort of overview of average or expected travel times between points (activities) in the system. Without such an overview it is very hard to reason about travel times that are well below 1 minute. For instance, if a step took 30 seconds whereas 10 seconds is the normal time, this would probably not be regarded as an issue if the normal time is not known.

- If we discover a very long and sequential model it might be better to simply look at the traces of specific case instead. For instance, a model like in Section 7.3.4 provides hardly any advantages over simply looking at a trace in a table.

- Analyzing a tray and pallet side-by-side (in one model) is not always necessary. If we want to see the handover from pallet to tray it is not necessary to keep the events for one of the objects after the handover. Therefore, for instance leaving out the pallet process after the products are put in the tray can be perfectly fine. This is the case unless we specifically want to analyze the last few steps of a pallet exiting the system, or unless the de-stacking comes to a halt because the empty pallets can no longer exit the system, which has a very slim chance of occurring. Hence, using a projection or selection to obtain models that hold only specific events is often a good practice.

- Using the lowest level case identifier (tray) and enriching the trace to include events of higher level identifiers, and then using a 1:1:1 case identifier works well to obtain event logs and process models that can be analyzed to answer specific questions or to find improvement potential

- Analyzing 1:n relations is best done in an artifact-centric manner, which we cannot obtain for now. We will propose a way to obtain this in Chapter 9.

In the next chapter, we distill these findings into a methodology for applying process mining on material handling systems with multiple case identifiers.
8. THE METHODOLOGY

In this chapter we briefly generalize our findings of the previous chapters, in particular Chapter 7, by specifying a methodology which can be used by engineers (and researchers) to apply process mining on material handling systems with multiple case identifiers. We create this methodology by abstracting from the details we provide in the previous chapters and by focusing solely on the steps taken to achieve the results. As such, we provide a high level overview of the steps to take to be able to analyze process models of other material handling systems (with multiple case identifiers). For Vanderlande we provided a more in-depth manual that includes the exact steps and queries to take/run to obtain event logs and do analysis. This chapter is split up in a section per step in the methodology. With this chapter we answer part of the fourth research question:

(IV) Given the insights in applicability, and the limitations of the current process mining landscape, we would like to distill a methodology for applying process mining on MHSs and we would like to identify future research problems.

“Identifying future research problems” is addressed in the concluding Chapter 9.

8.1. Specify Lifecycles

In Chapters 3 and 4 we reasoned that to properly link together the various objects in the system and to scope the cases, we need to define lifecycles for each of the objects. More specifically, we need to define a start and end point for each object, for each sub-process/use case in the system that we want to analyze. To find and define these lifecycles we apply the following steps:

1. Obtain domain knowledge:
   a. Read documentation, or create missing documentation together with experts
   b. Talk to/interview experts
   c. Go visit the system you study in real life
   d. Familiarize yourself with terminology
   e. Get to know the processes

2. Draft models by hand
   a. Model the separate modules (if they exist), or model sub-processes
   b. Model the interaction of these modules/sub-processes based on 1 object, for each object
   c. Combine the models for each of the objects to obtain a complete process model, showing the interaction between the objects and sub-processes

3. Evaluate models with various experts
   a. Evaluate with domain experts for correctness and applicability
   b. Evaluate with modeling experts for model soundness and usability
   c. Iterate steps 2 and 3 until satisfied – preferably finishing with models that are very close to the real system

4. Use models to define start and end for lifecycles (per object)
   a. Define specific places or activities that form the start or end of a lifecycle for an object, for each object, for instance using a projection on higher level process steps

8.2. Define a Log Extraction Specification

For event log extraction we need a specific input that defines a ‘case’ and supplies input for the log extraction itself. We define this log extraction specification by specifying case time-frames and a case identifier, by applying the following steps:

1. Specify case time-frames
   a. Explore data to find which columns/events correspond to start and end of lifecycle (per object)
   b. Use unique identifiers (object ID), or other information, to join these start and end events together to obtain a single table (or other fitting data structure) that holds the object id and the start and end time for that object – for the specified lifecycle (depending on the sub-process/use case)

2. Specify a case identifier
   a. Investigate the relations between objects: is it a 1:n relation? Then follow steps b and c, if not step b can be skipped.
b. Enrich trace towards object with finest granularity (in our case a tray), i.e. the object at the "lowest level"

3. **Use the case time-frame and case identifier to specify the log extraction specification**

### 8.3. Log Extraction Specification Extraction

We followed an ETL approach as there is not a one-button-press solution yet, but we define a method for log extraction: first specify, then implement. Hence, we split the problem of log extraction into: 1) creating a log extraction specification (LES) that maps database tables and columns to a case notion, and 2) a generic algorithm for log extraction that extracts a log for a given LES. Creating the LES itself is quite extensive and extracting them by hand for each case would be very cumbersome. Therefore, we define a LES extraction query/algorithm to do this for us. We still have to manually create or write this algorithm, as the specifics of the database and system itself determine the concrete query that needs to be written to obtain the log. As a general guideline we have created an abstract ‘algorithm’ that outlines what each step of the actual algorithm or query should do, as follows:

**Input:**
- Start event type and End event type (per object type)
- Timestamp columns
- Case identifier columns (Object ID columns)
- Task ID columns (or likewise)
- Any other columns that can help with linking together events
- Joining logic (i.e. first occurrence after start is end)

**Steps:**
3) Per object do the following
   a. Select start events
   b. Select end events
   c. Join the start and end events on Object ID (potentially using additional operations to define case timeframes for objects with persistent IDs)
4) Join the Object case time-frame tables one by one

**Output:**
✓ A Log Extraction Specification

### 8.4. Log Extraction

Actual event log extraction is quite straightforward once a LES has been created, and for that reason we have defined a generic log extraction algorithm as below. This high-level algorithm can be fit to any specific database system by defining the corresponding queries for retrieving objects (case identifiers) and event timestamps from the tables in the database.

**Input:** Log Extraction Specification (Table)

**Algorithm** `extractEventLog(LES)`:
```
1   eventLog = new List()
2
3   for each row in LES:
4       CaseId = concatenate(ObjectId1...n)
5       for each ObjectId column:
6           event_records = Query for ObjectId between StartTime and EndTime
7           for each Event in event_records:
8               Create Activity from some Event.Attribute
9               eventLog.append(Activity, Event.timestamp, CaseId)
10          export eventLog.csv
```

**Output:** Event Log
8.5. Analysis

Finally, to analyze the obtained event logs there are a number of tools that can be used to discover models and analyze them. We list the three main tools and their advantages and disadvantages:

1. **Disco**
   - Easy to use freemium industrial tool
   - Quite powerful filtering
   - Nice visual performance analysis
   - Option to animate log over model
   - Given careful pre-processing (and/or use case selection) can work really well
   - Discovers a directly follows graph (DFG), and thus does not handle parallelism
   - Does not allow to adjust/select case id, timestamp, etc. when loading XES files
   - No possibility to readjust visualization by hand
   - Often results in so-called spaghetti models (due to DFG)
   - Filtering is good but does not allow for (very) specific projection

2. **Performance Spectrum Miner**
   - Way to obtain and visualize correct performance information
   - Quite easy to find recurring patterns
   - Allows for blacklisting or whitelisting parts of model (so it provides filtering)
   - Might be a bit difficult or too in-depth for an engineer
   - Still in research phase, GUI needs updating to increase user-friendliness (for industrial use)
   - Creating a useful/logical segment ordering is a lot of manual work and requires a lot of domain knowledge

3. **Interactive Data-aware Heuristics Miner**
   - C-nets work very well for parallelism
   - Lots of options to adjust/finetune model
   - Causality analysis might be interesting for finding deviations
   - Lots of options to adjust/finetune model
   - Might be a bit difficult or too in-depth for an engineer
   - ProM is very research focused and thereby not the most user friendly tool

Depending on the analysis question using a 1:1(1):1 case identifier often works well to find issues, outliers, or to do performance analysis. To further finetune the models we have identified a few pre-processing steps to be executed either inside the analysis tool or in a separate framework or environment. We present these steps per main challenge of the process models for material handling systems with automated batching and multiple case identifiers (objects), which are parallelism, extremely large models with highly variable traces, divergence, and convergence.

- **Handling large models:**
  - If different activities portray the same type of activity occurring at different locations, consider aggregating these physical locations into hierarchical groups (or 1 activity)
  - Consider aggregating sub-processes into a one event, which then substitutes the entire sub-process
  - Projecting the event log to only contain 1 object
  - Analyze smaller use cases (sub-processes) at a time

- **Handling complex or meaningless models that do not show any relevant flow (due to parallelism):**
  - Project the event log to only 1 object
  - Project the event log to only 1 object with some very specific added events for the other object
  - Use the iDHM if possible, preferably with not too large models
  - Depending on variability: Inductive Miner (or even alpha miner) could also be used. If the trace variability is too large the inductive miner will often return a flower loop at some point in the model, which makes the model less useful.

- **Handling nodes with a high degree of in/outgoing edges and many meaningless loops (due to divergence):**
  - Add context to events, for instance by numbering them, or by taking specific information that is available
  - Removing events that suffer from divergence when possible (e.g. the Tasks which are not necessarily needed for all analysis questions)
• Handling unexplainable/wrong paths, skewed statistics, and many self-loops (*due to convergence*):
  o Split the objects which have a 1:n relation with each other into separate logs
  o Use artifact-centric process mining tooling if/when it becomes available
  o Take the self-loops for granted and realize that they are supposed to be there
    ▪ Extra care is needed to make sure *actual* loops are not overlooked
  o Use 1:1 case identifiers if duplication of events is not a big deal for the analysis goal, e.g. if we simply want to backtrack an object through the system the duplication is not too much of an issue, whereas if we are interested in statistics we should avoid duplication as much as possible.
In the research done for this master thesis we have exploratively analyzed process mining for material handling systems with multiple case identifiers, or more specifically warehouse automation systems with automated batching. We recall the research goal we identified in Chapter 1:

*Given data of a material handling system with automated batching and multiple different case identifiers, we would like to devise a methodology so that engineers (at Vanderlande) can apply current process mining techniques to do analyses.*

Adhering to CRISP-DM we split the research problem up into 4 research questions, which directly relate to each contribution of this thesis as we elaborated in Section 1.4. Moreover, we introduced four quadrants of process mining: Information handling systems with single and multiple case identifiers, and material handling systems with single and multiple case identifiers; the latter being the one we focused on in this study. We argued that an explorative study was imperative as this quadrant had not been studied before. However, before exploring the research questions, we first studied the solutions that are present for the other three quadrants together with some preliminaries: process mining in general, available tooling, ETL, batching, the Performance Spectrum Miner, and artifact-centric process mining. We highlighted where these approaches could be of use and why some of them cannot be used, emphasizing the similarities and differences to artifact-centric process mining.

Our first contribution is a conceptualization of warehouse automation systems, which includes conceptual process models and a description of how such a system operates. We identified the various identifiers in the system which could be used as a case identifier and devised so-called lifecycle models for the two objects in the system: pallets and trays. Using these models we identified lifecycles of the objects in the system and specified use-cases together with the corresponding lifecycles.

The next contribution is the description and extraction of a Log Extraction Specification. We defined case timeframes and a case identifier to constrain events to a case based on the lifecycles of the objects. We then outlined a high-level algorithm for extracting the LES in a query-like fashion. Using this LES as an input we defined a generic algorithm for event log extraction.

With the event log extraction algorithms we extracted several event logs from a Vanderlande system. Initial data exploration led to a clear view of the challenges: the number of activities in the models, the number of (meaningless) edges, the parallelism between tray and pallet, and divergence. We then proposed several improvement proposals like: aggregation, renaming, relabeling, and projection or selection. We analyzed the extracted event logs in a combinatorial study, where we examined the various improvement proposals and a selection of relevant tooling. We evaluated whether each of the proposals and tools were applicable on this domain of process mining and highlighted the limitations of the current process mining solutions.

Following up on the analysis we distilled a methodology for applying process mining on material handling systems with automated batching. This methodology covers the steps that need to be taken to obtain an event log and discover a process model to analyze. We explain each of the steps on a high level, providing a guideline for others to work with. However, the methodology still includes a lot of manual work and requires the researcher or engineer to have or gather extensive domain knowledge on the system. We provide guidelines to follow, but we do not yet provide tools or an implementation to reduce the required work for others. Nonetheless, the methodology does set up a clear step-by-step guide, which can be followed, and inspiration can be taken from the provided examples in this research.

The final contribution is an overview of the limitations of the current process mining solutions for the quadrant we studied. The main challenges for process mining for material handling systems with automated batching are as follows: automatically defining a LES and extracting event logs, parallelism between the objects, extremely large models with highly variable traces, divergence, convergence, and discovering models with 1:n relations in the case identifier. From the limitations and challenges we identified specific research problems for future work and propose a few solution ideas or approaches, which we present directly hereafter. We first go over some general limitations, then discuss the log extraction, and finally we discuss the analysis. We end by proposing a few visualizations or discovery algorithms that could help with analyzing the 1:n relations in the data.
Given data of a material handling system with multiple case identifiers, we would like to develop a framework that is able to automatically identify/discover the lifecycles of multiple (interacting) objects, and uses this information as an input to extract event logs.

Thus, the framework should be able to define a Log Extraction Specification (object lifecycles) and then extract this from the data, followed by the extraction of the actual event log. However, as the required inputs for such a framework are likely very specific per system it may be extremely difficult to automate or generalize. Thus, the framework should be able to define a Log Extraction Specification that is able to semi-automatically discover the lifecycles of multiple (interacting) objects, given a set of input parameters, and uses the resulting log extraction specification as an input to extract event logs.

Hence, we want to develop (or extend) a framework like XTract(2) [34] [11], but then for material handling systems with multiple case identifiers instead of information handling systems, as we were unable to apply it on this data because XTract was designed with different assumptions in mind. To (semi-)automatically find the start and end events of a lifecycle we could look for the first occurrence of an event, do that for a set of events (possibly defined by engineer), and compare which event occurs most frequently as a starting point. However, if all objects have a clearly defined lifecycle through their identifier, the only remaining challenge is linking objects together by automatically finding the link in the data (i.e. some event could hold both IDs).

Model Analysis
The analysis we carried out for this thesis is mostly limited to analyzing models with a 1:1 case identifier. This is due to the fact we cannot really analyze 1:n relations properly using the current process mining tools that can handle logs with a single case identifier only. However, we argued that analyzing 1:n relations does not necessarily provide new or required insights in a practical setting. Furthermore, we mostly limited the analysis to existing tooling and solutions. In the future the analysis could be extended to involve a specific miner or visualization for material handling systems with multiple case identifiers. We can use the definition of the Log Extraction Specification we
defined in this research as a guideline, as this already defines the links (i.e. interactions) between objects in a certain time-frame. Hence, we define a research problem to do develop a miner/visualization as follows:

*Given an event log that contains data of a material handling system with multiple case identifiers, which holds information on the lifecycles of each object and their interactions, we would like to develop a discovery algorithm and visualization that presents these lifecycles and interactions in a dynamic manner, such that data convergence no longer occurs.*

More specifically, we want this miner/visualization to be able to do the following (or at least the first):

- Discover the interactions between the lifecycles and model these like an artifact-centric process model
- Let the user decide what (and how) to aggregate into subprocesses
  - i.e. Define a hierarchy of sub-processes and use this to dynamically expand or collapse parts of the model
  - For instance show “aggregation levels” to go up or down for the entire process
  - For instance have a + or – on each ‘box’ containing activities to indicate it can be (de)aggregated
- Change the layout manually (i.e. by dragging activities)
- Switch between a non-aggregated version of ‘modules’ and an aggregated super-module
- Let the user specify physical aggregation through renaming as an input for the discovery algorithm

Furthermore, we mentioned that the Tasks and actual physical flow happen almost completely independently from one another, which leads to a high variability in the traces. To solve this without removing the Tasks, we propose to develop a discovery algorithm that can discovering a synchronizing model that synchronizes the separate ‘Task Model’ and ‘Physical Model’. The resulting model could for instance be a synchronized Petri net, or a BPMN model with messages as the synchronization.

As a last suggestion, to handle the large variability in the traces and to be able to specifically backtrack the process for one case, we propose to develop a discovery algorithm that works like the artifact-centric algorithm we propose above. However, instead of discovering the lifecycle for all objects, we want to glue together the trace models of the inbound pallet, followed by the trace model of a tray, and finally the trace model of the outbound pallet. In this way building a sort of reversed tree structure that allows an analyst to exactly follow the process of each product that was placed on an outbound pallet, given a set of identifiers or only the identifier of the outbound pallet as input. Again, the LES as we specified in this research can be used to determine which traces to glue together. Thus, the next steps mainly revolve around generalizing the results found in this research and developing a framework which can semi-automatically extract event logs and discover (artifact-centric) process models, based on the ideas we presented.
REFERENCES


APPENDIX A: BPMN AND PETRI NET MODELS

Note that this was initially part of Chapter 4, and as such may refer to concepts explained in Chapter 3, or point forwards to later proposed solutions in Chapter 4.

To find the artifact models, or to find the lifecycle of each module, each of the 5 modules as described previously have been modeled. Some new messages or activities have been introduced to cover the actual work of a module. These will be explained for each module separately, together with a discussion of the operation of the module. The corresponding BPMN and Petri net models will be elaborated and compared to find the best fit.

De-Stacking
As explained, de-stacking is the process of filling trays with products from pallets (step 4 - Figure 8). This is done in steps: first the products are removed from the pallet layer by layer, after which they are put in a tray one by one. The maximum capacity of a tray depends on the size of the product but in general the capacity is between 1 and 4 products. Empty trays are routed towards the de-stacker (outside of the de-stacking process scope) and wait until they are used at the de-stacker. Hence, we assume there are always enough empty trays to be filled by the de-stacker. As such, we do not have to model the buffer of empty trays and can disregard the control process that belongs to it. Figure 43 shows both the BPMN model and the Petri net.

The BPMN model is quite straightforward, but can only be executed when data is accounted for. If this is not the case, it is not sound as the AND-split would be taken multiple times leading to an unbounded number of tokens on “Put in Tray”, which are not synchronized at the AND-join. The process starts when a task is assigned, after a demand for a certain product is present. The task is started by a de-stacker and it removes the first layer from the pallet. The decision at the XOR-split is based on whether this was the first time getting there (i.e. if it is after the removal of the first layer). If so, continue to an AND-split which splits the process in a part for the pallet and a part for the tray. Going upward is the pallet flow, which reaches another XOR-split. This time the flow is continued based on the state of the pallet. If the pallet is empty go back down to the AND-join, if it is not empty, remove another layer and go into the loop again. It must be noted that now the ‘first?’-XOR-split will always result in NO, and as such there remain only 2 parallel flows. The tray flow, downward from the AND-split, is even simpler. Products are put in a tray, after which a XOR-split the decision is made whether the task is done or not. If not, simply loop back to putting products in trays. If the task is done go to the AND-join. Here the tray and pallet flow come together again after the task is done. Then finally a Task Complete message is sent and the process ends. Hence, during the process multiple “remove from pallet” and “put in tray” messages are sent. “Put in tray” is assumed to only be sent once for each tray, and “remove from pallet” is assumed to only be sent once for each layer on the pallet.

The Petri net is quite simple as well, but uses edge weights and inhibitor arcs to bound the process a bit more precisely to the number of layers removed from a pallet and the number of trays filled during a task. The task starts after it is assigned. The Task Assign message will also send the correct number of tokens to both counter places for layer and trays (in the example 3 and 4). Again, after the first layer is removed (which is modelled explicitly as a separate activity here), the process splits into a parallel flow between pallet and tray. Here, layers are removed and trays are filled exactly as often as the number of tokens in the counter places. The task can only be completed if both counters are empty, and if there is a token in both the ‘buffer’ places above remove player and put in tray. This ensures proper completion of the (workflow) net.

Both models are very much alike in terms of operation. However, the Petri net is slightly more precise given its exact limits instead of decisions made on XOR-splits. Moreover, the Petri net only requires the Task Assign message to include the correct number of layers to be removed and trays to be filled, whereas the BPMN model needs constant regulation of the state of the process as decisions have to be made multiple times. As such, it is probably better to use a Petri net model for workflow enforcement. However, this is not what we are after. Both models can show the behavior of the de-stacker but are fairly complex for unexperienced analysts. Furthermore, they do not provide clear start and end points for the lifecycle of either a tray or a pallet. Therefore a need for a different kind of model arises, which is discussed in Chapter 4.
Stacking

Order fulfillment is achieved by combining multiple different products on one pallet that will be sent to the shop. This is done by the stacker module, which takes products from trays one by one, and then puts them onto the pallet one by one. All trays have to be present and at least on their way to the stacker before a task is started, and as such it is assumed that the next tray will always be there. Moreover, it is assumed that there is always an empty pallet that can be used to stack onto. Lastly, empty trays are not specifically handled by the stacker, and thus left out. This eliminates the need to model the empty pallet buffer and the trays arriving and leaving at the stacker.

Figure 44 shows the BPMN model for stacking on the left. It is a very straightforward model with a single loop. The process starts when a stacking task is assigned as a result of a store order coming in. The task starts when the first tray arrives (not modeled). Products are then taken from a tray and subsequently put onto the pallet. Then a XOR-split based on whether or not the task is done determines the flow. If the order is not done the process loops back to taking products from a (new) tray. If the task is done, the task complete message is sent and the process terminates.

The Petri net is a bit more elaborate, as it accounts for the exact number of trays to be used by the stacker. Again, edge weights and inhibitor arcs are used to do so without creating an unsound net. The process starts by assigning a task, this also puts the correct number of tokens into the counter place—this case 4. The task starts when the first tray arrives at the stacker, which is not modeled as with the BPMN model. The products are taken from the tray and put on the pallet, after which the ‘Next Tray’ is taken and the process loops back to the place before ‘take from tray’. This loop continues until the counter place is empty, after which ‘Next Tray’ can no longer fire and ‘Task Completed’ is finally activated and shall hence fire. The process can thus only end when all assigned trays have been used for stacking.
Again, as with de-stacking, the models very closely resemble each other. The Petri net also still requires less observing and only requires a decent input, whereas the BPMN model requires constant monitoring for the XOR-split. However, as is the nature of both modeling languages, the models are not trivial enough for our liking. Hence, a more simplified model is presented in Chapter 4. Nonetheless, it must be noted that the BPMN model is almost as simple as it gets, the only downside being the need for data to decide which way to go at the XOR-split.

Retrieve, Store, and Transport

The other three remaining modules, pallet/tray storage and transport, all function the same at this level of abstraction. As can be seen in Figure 45 the process is a simple sequence of tasks. A Petri net has not been made for this process as it would be exactly the same except there would be places in between the activities. In contrast to the models presented for stacking and de-stacking this model could be considered too abstract. It shows hardly any details, which might be interesting for the analysts. Hence, a ‘Location Status’ message will be introduced which covers a slightly more detailed version of retrieve, store, and transport. It will tell the system where exactly each tray or pallet is at a given moment in time. The locations will be limited to start and end points, like the entrance in the warehouse or the entrance to a storage facility or the stacker. This way the interactions between the modules can be shown in more detail. Moreover, it is important to keep track of not only what task the tray or pallet is undergoing, but also to see where it is in the physical system. Doing so, an analyst might be able to find imbalances or bottlenecks in the physical design and software of the system.
APPENDIX B: COMPLETE DISCUSSION OF RESULTS

Full Model – Numbering Context

Figure 46 shows a zoomed out process map of the discovered model for an event log with added context in the form of numbering. The shown model is set to 0% paths, hence only showing the main flows. We obtained this model by taking the full event log and adding a simple count to each activity, like we explained in Section 6.3.2. We then loaded the newly obtained event log directly into Disco.

The model is very lengthy and has a lot of sequential paths. However, this is expected behavior as we removed most loops from the event log by adding a counter to each activity, which should result in any specific activity only occurring once. We still observe a lot of possible paths to follow, because as we explained, the context of an activity with an added “_1” does not have to be the same as another activity with “_1”. Nonetheless, if we filter this model to show us 1 case we get a very good overview of what exactly happened (because we are then looking at a trace model). So we can use this particular type of model for specific analysis on 1 case. However, a model like this is not so useful for large scale analysis as it still requires a lot of work to walk through one instance of such a model. Moreover, parallelism between pallet/tray still exists so we still have false edges, which we have to be careful with when analyzing. Though, if enough domain knowledge is obtained, one should be able to distinguish these false edges quite easily.

Hence, the model obtained by adding numbering as context is useful for one analysis question but does not serve a larger purpose. As such, this is also the only result for this type of log we show.
Figure 47 shows the full model we discover when we add context in the form of specific information. In this case we have used the destination of a Task to add context to the activities. The model is set to 0\% paths, but is still quite a so-called spaghetti model, and we cannot really make any sense of it. However, there is a nice thing happening. See the cut-out below in Figure 48, where the process clearly splits up into 3 ‘paths’ which eventually join again. This is a promising pattern, which clearly shows how the context can improve a model by showing more specific information and by removing redundancy. However, on such large models it is a bit lackluster, as there are still many edges surrounding the pattern. Moreover, if we solve the issues of the large models, we do not really need to add context anymore, as the models are already clear – as we will see later in this chapter.
Figure 49: "High level" order model, 0% paths (left) and 100% paths (right)

Figure 49 shows 2 models that only show Task events, in this case called Orders. On the left we show the 0% paths setting, which shows a very clean model with a clear execution order of tasks. On the right we show the 100% paths setting, shows a slightly more cluttered model, indicating there are a few orders in which a sequence of tasks can be executed by the system. Some of these differences will be due to sub-second ordering or simply because for instance sometimes the system first assigns a transport task and then the storage task, while sometimes it does so vice versa.

The models show a nice overview of the follow up of tasks but are not necessarily useful for finding specific problems. Therefore such models should mainly be used to analyze the general flow of the system on a high level, and not for troubleshooting for specific incidents. The models can potentially show deviating process flows, especially since the 100% paths model is still legible; in this case we did not find any deviating process flows.
Order Model – Numbering Context

Figure 50 illustrates how many tasks are sent or can be sent for 1 full execution of the process. Especially in the 100% model on the right we see many possibilities to skip entire parts of the process, indicated by the many edges on the right side of the model ("skips"). Other than that these models are not too useful for analysis purposes, but do offer a simple overview of which tasks were executed in what order, also for specific cases. However, counting tasks can also easily be done using a query on the database or using a script on the event log, and thus we do not necessarily need this version of the order model. Nonetheless, it provides a quick understanding of exactly how many tasks are executed for one execution of the full process.
Figure 51: Order model with specific context added (destination)

Figure 51 shows the order model but with specific context added. Again, we used the destination of orders to add the context. In this case we show the 0% paths setting, as the 100% paths still lead to a so-called spaghetti model. The model shows a bit more specified information, and allows to follow the orders/tasks on a more physical level. For instance, at the end it shows a split into three different destinations, which could help with analyzing the load balancing on the system. However, analyzing such a model requires a very good knowledge of the physical layout of the system to be able to draw any conclusions. Moreover, for analyzing load balancing in the system a physical flow model can be used as well, which shows more information and is thus probably better to use for that case. As such, these models do not provide too much extra in comparison to other models we can obtain.
Figure 52 illustrates what happens when we project the event log into a log that only holds 1 object, in this case the inbound pallet. The model set on 100% paths, thus showing the entire physical flow, but is still very compact. As such, it is a very good and useful model, though this is to be expected as we have a single case identifier here (the pallet ID). We obtained this model by using the filter function in Disco to filter in the events regarding the pallet, using the physical flow log as a starting point.

As an addition, we have added the general ‘location’ in italic to the model. We clearly see that the process is followed exactly as described in the models we created in Chapter 3 and 4. However, there is an unexpected loop back towards the receiving. After investigating this with an expert we found that this loop only occurs for empty pallets that they used for testing the flow capabilities of the system, and as such this loop should not occur when the system is actually running daily operation.
Specific Pallet/Tray Projection (De-Stacking)

Figure 53: De-stacking model of pallet and tray (projection)

The model in Figure 53 shows the handover from pallet to tray at de-stacking, at 100% paths. First, pallets are retrieved and brought to de-stacking, where layers are removed (dark blue activity with self-loop). Then, items are put on the tray, which is coincidentally the start of the trays lifecycle. The tray then goes towards storage, where the lifecycle for the de-stacking use case ends per definition. We have filtered out the pallet events after the handover to reduce the issues that arise from the parallelism between multiple objects. The resulting model can show potential problematic delays in bringing pallets towards de-stacker, which can potentially result in a delay at stacking. As such, using very specific selection of events to keep only those that we are interested in for a specific question, we can create a model that allows us to analyze de-stacking as a whole, without removing information that could be of interest. The only downside of filtering out the exit of the pallet (i.e. after the handover) is that whenever something goes wrong here we will not see it. However, this part of the system is very simple as it is just an outfeed, which can also have redundancy to deal with potential breakdowns.

We obtained this model through very specific selection of events to keep, using the filtering options in Disco after extracting an event log for the de-stacking use case. Doing so requires substantial domain knowledge as we need to know exactly what can and what cannot be left out. Hence, these models can be very useful for analysis but require the engineer of analyst to know everything about the system and data, which can pose an obstacle.
Tray Projection

Figure 54: De-stacking to stacking model of tray only (projection)

Figure 54 shows the projection of a tray that goes from de-stacking to stacking, at 100% paths. It is a decent model, which is to be expected again as we have a single case identifier (the TrayID). We can analyze a model of this size quite well, but layout is sub-optimal and cannot be changed. As the trays first go into storage, are then retrieved, and finally go back to storage (or empty tray storage), the storage activity is placed a bit awkwardly in the model, resulting in some edges going through the model and seemingly looping. This is fine when an engineer can reason that the 2nd time the tray reaches the storage it should be the end of the process. If this is not the case, we could opt to add some context to the storage activities only by numbering them.

We obtained this model through filtering in Disco, just like with the pallet before. However, we could also opt to extract an event log that only queries for the tray events. This is a choice to be made: do we want a full log and from there select/filter/project into smaller logs, or do we want to extract a number of logs that are already smaller. Depending on the querying language the latter is probably easier, or at least less time consuming for the engineer, whereas the first results in extra log extraction time. We could always do both if we have enough processing power and time, that way providing multiple logs for analysis, which potentially require less pre-processing.
The model in Figure 55 is a high level aggregation of the process. We have added the related locations as we described in Chapter 3. In essence such a process model shows the steps we illustrated in Figure 8. This high level model is nice to get a high level feel for what the system does, and how it is built up. For instance, an engineer could start at this level, and gradually work his way downward to a full physical flow model. However, the model is rather pointless for analysis as we cannot dynamically change the aggregation level, and at this level there is not much we can say or find.

Nonetheless there are 2 interesting edges in this model, which we highlighted in red. The edge from “Deslaver” (the pallet exit) towards “Posisorter/Traystore” (a sorter before the tray storage) is false. No pallet can physically make this move, and as such the edge is a result of the parallelism between the pallet and tray, like we explained in Chapter 4 and 6. As for the edge from “Deslaver” towards “Receiving”, this is the same as in Figure 33, where empty pallets are sent around for testing purposes. Hence, the model can bring potential problems to light, but is quite limited in its capabilities to do so.
The model in Figure 55 is a bit too high level to be able to easily draw conclusions. As such we created another aggregation level, which is shown in Figure 56. The model shows the process on a slightly lower level, hence making it a nice step in between the highly aggregated model and the full (physical flow) models. For instance, the model can be used to find bottlenecks at a relatively “medium” level, i.e. in between ‘De-stacking’ and ‘De-stacking module 2, tray exit’. This is a good level as it provides enough detail to find the initial problem, and does not require too much domain knowledge. The lower level investigation could then potentially be done by an engineer with more domain knowledge.

However, as an example of the difficulties/challenges for aggregation, we highlighted a part in red. This reads: Reading, ReshuffleLoop, ReshuffleEntry from top to bottom. The ‘Clearing’ is connected to the inbound pallet part and to the reshuffle loop/entry for a tray. This is because both the pallet and tray conveyor setups have an overflow mechanism that directs the object towards a ‘clearing’ station where manual investigation must take place. Due to the aggregation these stations have become one, hence mixing up the pallet and tray flow unintentionally. Therefore, extra care has to be taken when aggregating activities, as some may have identical names whereas they are different in reality. Furthermore, it would be nice to be able to dive deeper into certain parts by expanding them through dynamically changing the level of aggregation – something that cannot be done in Disco (and more tools).
Stacker Instance Comparison

The analysis in the following section aims to answer a specific question: “Can we detect any differences in how instances of the same module operate?” for the first model, and “Can we find the inserted performance issue?” for the second model. As there are multiple instances of one module, in this case stacking, it might be interesting to analyze if there are any differences between them, both in terms of flow and performance. The data is from an emulation of the actual system, where we manually inserted a temporary failure. We extracted this data and filtered it down to only contain the tray events for the stacking modules. Moreover, we did not aggregate the instances together as we specifically want to see them in the analysis.

Figure 57: Comparison of 3 stacker module instances

Figure 57 shows three stacker modules next to each other. The process starts when trays are on their way to the module, and ends when they exit the stacker, either towards storage or to the empty tray storage. We see that the module on the left has a slightly different process map. Each module has the ‘stacking’ flow, which has the broadest/darkest edges, and three exits: 2 storage aisles, and 1 towards empty tray storage (often with a darker edge). The module on the left does not have these three exits, but only two. However, it is not operating differently, the missing path just has not ‘happened’ (yet). If this would remain the same for a very large data set this could indicate something peculiar is going on with how the system distributes trays between the two connected storage aisles.

Figure 58: Excerpt of the comparison of 3 stacker module instances, in performance view

Figure 58 shows the same model as in Figure 57, only this time it shows the performance view of only one of the three modules (the middle one). At a first glance we immediately find the apparent bottleneck, indicated by the broad red edges. We see that the ‘PalExit4008’ and ‘PalExit4007’ (exits of the stacker module) have a long time before happening; the same goes for ‘TrayExit1’. This indicates that the exit of the stacker is somehow blocked or slowed down. This is also exactly what happened: in the emulation we temporarily disabled the exit of the module. The result is a build-up of trays that cannot exit the module, and we thus see a delay at the edges going towards the exits: PalExit4008, PalExit4007 and also at TrayExit1 (though to a lesser extent as it is further back in the module). Disco also provides an animation of the traces over the model. We show a screenshot of this in Figure 59, which also clearly shows the delay happening (especially when playing). Once we remove the failure, the build-up of trays resolves and everything goes back to normal again.

Hence, we can conclude that we can do very specific analyses on the system by extracting specific logs from incident days (or hours) and filtering them down to where we know the incident happened.
Figure 59: Animation on the model also clearly shows the delay (when playing)
To deal with the storage locations for performance analysis, we propose to split the logs in Section 6.2. We show only the ‘before storage’ part of the tray model in this case, as the other models (before/after storage for pallet and tray) lead to the exact same conclusions. All ‘half’ models are obtained by first projecting an event log that only includes the tray (or pallet) from the physical flow log, or by extracting an event log that only includes the tray or pallet. Then, we run a simple script to split each trace right at the “stored” activity. The ‘Before’ log contains the “stored” activity, whereas the after does not (because then it would still show up as the bottleneck).

The resulting models, like the one in Figure 60, make performance analysis a lot easier, as now we can quickly see other potential bottlenecks instead of only the storage waiting time. However, this only provides an indication of what might be a bottleneck, or a part of the system operating slower than it is supposed to. Because most ‘travel times’ between activities (locations) are quite small (i.e. <1 minute) it is hard to estimate whether or not a delay actually occurred. To do so, we need an overview of average times between segments/parts in the system to make any conclusions. Without such an overview we can never conclude that 20 seconds is ‘too slow’, and that something must have been wrong as it was supposed to be only 10 seconds. Hence, splitting the log in half and then discovering a model makes performance analysis easier, but to properly do so we need more information.