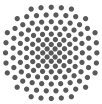

Universität Stuttgart

Metadata Management in Virtual Product Development to Enable Cross-organizational Data Analytics

Von der Graduate School of Excellence advanced Manufacturing Engineering
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Julian Ziegler

aus Ludwigsburg

Hauptberichter: Prof. Dr.-Ing. habil. Bernhard Mitschang

Mitberichter: Prof. Dr.-Ing. Angelo Brayner

Tag der mündlichen Prüfung: 3. September 2024

Institut für Parallele und Verteilte Systeme (IPVS) der Universität Stuttgart

2024

Acknowledgements

This thesis is the result of a doctoral project conducted at the Graduate School of Excellence advanced Manufacturing Engineering (GSaME), in collaboration with the Application Software department of the Institute for Parallel and Distributed Systems (IPVS) at the University of Stuttgart, and MANN+HUMMEL GmbH in Ludwigsburg.

First of all, I would like to express my sincere gratitude to my doctoral advisor, Prof. Dr. Bernhard Mitschang, for his intensive and constructive supervision over the past years. I am also very grateful to Prof. Dr. Angelo Brayner for taking on the role of co-examiner.

I would like to thank Dr. Peter Reimann, junior research group leader at the GSaME, for his continued support and guidance throughout my doctoral project. My appreciation also goes to Dr. Thomas Ackermann and Dr. Gabriele Erhardt for their valuable advice and organizational support within the GSaME.

I am deeply thankful to my colleagues at the IPVS for their daily collaboration and for the valuable insights they provided. In particular, I would like to thank Dr. Corinna Giebler for being the best office mate, Dr. Michael Behringer for his steadfast presence even late at night, Dr. Christoph Stach for his well-typeset knowledge in all things (except sushi), Sarah Oppold for many productive writing sessions, Yannick Wilhelm for countless feedback sessions, and Dr. Alejandro Gabriel Villanueva Zacarias for bringing fresh perspectives. I am also grateful to Dr. Manuel Fritz, Dennis Przytarski, Daniel Del Gaudio, Dr. Christian Weber, Rebecca Kay Eichler, Dr. Pascal Hirmer, Frank Steimle,

Dennis Treder-Tschechlov, Prof. Dr. Holger Schwarz, Dr. Vitali Hirsch, Marco Spiess, and Eva Strähle. Additionally, I would like to acknowledge the contributions of several students from the University of Stuttgart: Florian Niebling, Tobias Boceck, Rudolf Münch, Anatoli Steinhauer, Denis Moslavac, Paul Bredl, Janek Greif, Michael Erdemann, and Simon Weiler.

I would also like to thank my colleagues at MANN+HUMMEL GmbH for their contributions to my research. In particular, I appreciate the advice and guidance of Dr. Florian Keller, as well as the support from Dr. Kai Höss, Alexander Kilian, Dr. Christoph Schulz, Dr. Thomas Gose, Dr. Carolin Fink, and Dr. Martin Lehmann. An especially heartfelt thank you goes to Friedemann Hahn for his support and the increadible book cover design used for this publication. I am also thankful to Harald Kurz and Jürgen Lang for giving me the flexibility to complete my thesis.

Special thanks to Dr. Steffen Hirschmann, Sarah Jane and Roberto Fontanarosa, and Steve Brieller for their support. Moreover, I am very grateful to Ioannis Tilaveridis, Ying Xiong, Aleksi Riitala, Linming Pan, and Janne Gröndahl for ensuring I occasionally left the office to see the world.

Finally, I want to express my heartfelt gratitude to my family for their unwavering support and encouragement throughout my studies. I am especially thankful to my parents, Regina and Thomas Ziegler, my grandparents, and my sister, Jana Ziegler, for proofreading this thesis.

Contents

List of Abbreviations

A

ACID	*Atomicity, Consistency, Isolation, Durability*
API	*Application Programming Interface*

B

BPMN	*Business Process Model and Notation*

C

CAD	*Computer-aided Design*
CAE	*Computer-aided Engineering*
CAT	*Computer-aided Testing*
CFD	*Computational Fluid Dynamics*
CRISP-DM	*Cross-industry Standard Process for Data Mining*
CRUD	*Create, Read, Update, and Delete*
CSMM	*Core Scientific Metadata Model*
CSV	*Comma-separated Values*

E

ELT	*Extract-Load-Transform*

ETL	*Extract-Transform-Load*

F

FEA	*Finite Element Analysis*
FSM	*Frequent Subgraph Mining*

G

GEMMS	*Generic and Extensible Metadata Management System*
Graph'MeMo	*Graph-based Metadata Model*

H

HDFS	*Hadoop Distributed File System*
HoQ	*House of Quality*
HTML	*HyperText Markup Language*

J

JSON	*JavaScript Object Notation*
JT	*Jupiter Tessellation*

K

KDD	*Knowledge Discovery in Databases*
KRR	*Kernel Ridge Regression*

M

MEDAL	*Metadata model for Data Lakes*

O

OCL	*Object Constraint Language*
OLAP	*Online Analytical Processing*

P

PDE	*Partial Differential Equation*
PLM	*Product-Lifecycle-Management*

Q

QFD	*Quality Function Deployment*

R

RBF	*Radial Basis Function*
RDBMS	*Relational Database Management*
REST	*Representational State Transfer*

S

SDB	*Scientific Data Bag*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SVMs	*Support Vector Machines*

U

UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*

W

WS-Discovery	*Web Services Dynamic Discovery*
WSDL	*Web Services Description Language*

X

XCT	*X-Ray Computed Tomography*
XML	*Extensible Markup Language*

Abstract

Due to the advancing digitalization, companies are increasingly adopting computer-aided technologies. Especially in product development, computer-aided technologies enable a gradual shift from physical to virtual prototypes. This shift towards virtual product development includes design, simulation, testing, and optimization of products, and reduces costs and time needed for these tasks. Companies with strong activities in the field of virtual product development generate large amounts of heterogeneous data and wish to mine these data for knowledge. In this context, metadata is a key enabler for data discovery, data exploration, and data analyses but often neglected.

The diversity in the structure and formats of virtual product development data makes it difficult for domain experts to analyze them. Domain experts struggle with this task because such engineering data are not sufficiently described with metadata. Moreover, data in companies are often isolated in data silos and difficult to explore by domain experts. This calls for an adequate data and metadata management that is able to cope with the significant data heterogeneity in virtual product development, and that empowers domain experts to discover and access data for further analyses.

This thesis identifies previously unsolved challenges for a data and metadata management that is tailored to virtual product development and makes three contributions. First, a metadata model that provides a connected view on all data, metadata, and work activities of virtual product development projects. A prototypical implementation of this metadata model is already being applied

to a real-world use case of an industry partner. Based on this foundation, the second contribution uses this metadata model to enable feature engineering with domain experts as part of data analyses projects. Going further, data analyses can directly use the metadata structure to provide added value without having to access the large amounts of product data. To this end, the third contribution utilizes the metadata structure itself to enable a novel approach to process discovery for product development projects. Thus, process structures in development projects can be analyzed with little effort, e.g., to identify good or inefficient processes in development projects.

German Abstract

Aufgrund der fortschreitenden Digitalisierung setzen Unternehmen zunehmend computergestützte Technologien ein. Insbesondere in der Produktentwicklung ermöglichen computergestützte Technologien eine schrittweise Transformation von physischen zu virtuellen Prototypen. Diese Verlagerung hin zur virtuellen Produktentwicklung umfasst Design, Simulation, Test und Optimierung von Produkten und reduziert Kosten und Zeitaufwand für diese Aufgaben. Unternehmen, die stark im Bereich der virtuellen Produktentwicklung tätig sind, generieren große Mengen heterogener Daten und möchten diese Daten zur Wissensgewinnung nutzen. In diesem Zusammenhang sind Metadaten ein wichtiger, aber oft vernachlässigter Faktor für die Datenerkennung, Datenexploration und Datenanalyse.

Die Vielfalt der Struktur und der Formate virtueller Produktentwicklungsdaten erschwert den Domänenexperten die Analyse dieser Daten. Domänenexperten fällt diese Aufgabe schwer, weil solche technischen Daten selten ausreichend mit Metadaten beschrieben sind. Darüber hinaus sind die Daten in den Unternehmen oft in Datensilos isoliert und für Domänenexperten nur schwer zugänglich. Dies erfordert ein adäquates Daten- und Metadatenmanagement, das in der Lage ist, mit der signifikanten Datenheterogenität in der virtuellen Produktentwicklung umzugehen und die Domänenexperten in die Lage versetzt, Daten zu entdecken und für weitere Analysen zu nutzen.

Diese Arbeit identifiziert bisher ungelöste Herausforderungen für ein auf die virtuelle Produktentwicklung zugeschnittenes Daten- und Metadatenmanage-

ment und liefert dazu drei Beiträge. Erstens, ein Metadatenmodell, welches eine vernetzte Sicht auf alle Daten, Metadaten und Arbeitsaktivitäten virtueller Produktentwicklungsprojekte bietet. Eine prototypische Implementierung dieses Metadatenmodells wird bereits in einem realen Anwendungsfall eines Industriepartners eingesetzt. Auf dieser Grundlage nutzt der zweite Beitrag dieses Metadatenmodell, um Feature Engineering mit Domänenexperten als Bestandteil von Datenanalyseprojekten zu ermöglichen. Darüber hinaus können Datenanalysen die Metadatenstruktur direkt nutzen, um einen Mehrwert zu schaffen, ohne auf die großen Mengen an Produktdaten zugreifen zu müssen. Dazu nutzt der dritte Beitrag die Metadatenstruktur selbst, um einen neuartigen Ansatz zur Prozessfindung für Produktentwicklungsprojekte zu ermöglichen. So können Prozessstrukturen in Produktentwicklungsprojekten mit geringem Aufwand analysiert werden, um beispielsweise gute oder ineffiziente Prozesse in Entwicklungsprojekten zu identifizieren.

Chapter 1

Introduction

Due to the ongoing digital transformation, companies are increasingly relying on computer-aided technologies. In product development in particular, computer-aided technologies enable a gradual transition from physical to virtual prototypes to reduce costs and time needed for developing new products [VK11; HDGL13]. This allows for a progressive digitalization of product development and its main constituents: design, simulation, testing, optimization, and verification of products.

Most activities in product development are based on three categories of computer-aided technologies: CAD, CAE, and CAT [HDGL13]. Computer-aided Design (CAD) applications design a virtual prototype as graphical 2D or 3D models. Based on these models, Computer-aided Engineering (CAE) applications perform virtual product tests, i.e., they simulate and optimize product properties without the need for expensive physical prototypes. However, even with cutting-edge computer simulations, final verification still needs some amount of testing with physical prototypes [Cou+12]. These physical tests are supported and partially automated by Computer-aided Testing (CAT) applications.

Through the extensive software support, most activities in product development generate large amounts of data that are stored in Product-Lifecycle-

Management (PLM) systems. In addition, a wide range of competing software applications is available, which rarely meet open standards with regard to data formats. This leads to a strong heterogeneity of data formats, metadata, and interfaces for data access and data exchange. The resulting challenges in handling these data and metadata, as well as their subsequent use for data analyses, are the fundamental focus of this thesis. The following Section 1.1 motivates this thesis in more detail. Section 1.2 presents research challenges and Section 1.3 lists the respective contributions of this thesis towards these challenges. Section 1.4 depicts an outline of this thesis.

1.1 Motivation

Modern product development is increasingly virtual thanks to advancing digitalization and the adoption of computer-aided technologies. This reduces costs and time needed for developing new products [VK11].

Virtual product development covers multiple disciplines [HDGL13]. CAD allows creating products digitally and reduces the need for physical prototypes. CAD can be used in conjunction with other computer-aided software applications and even plan the manufacturing process of products. CAE describes the use of computer software for product simulation, optimization, and validation. This is often based on a CAD model of a product. Both disciplines reduce the need for physical product testing and, consequently, the cost and time for product development [VK11; HDGL13]. In CAT, computers help with testing physical prototypes and digitize one of the remaining physical steps in product development.

A wide range of software applications for CAD, CAE, and CAT tasks exists and is used in companies where they produce large amounts of data. Due to the diversity of these software applications and the different tasks they solve, the data they produce and consume is highly heterogeneous in their format and structure [HDGL13]. This makes it difficult to extract more value from these data via subsequent data analyses. Another difficulty is that product development projects involve the collaboration of multiple organizational units in a company. Usually, different organizational units store their data in their

own isolated data silos [CKC20]. This isolation of data further limits access and hinders cross-organizational data analyses by domain experts.

Data in virtual product development is often complex to understand and requires domain expertise to interpret properly. Viaene [Via13] suggests that domain experts should have control over the process for data discovery and exploration within data analyses, and that data scientists only guide domain experts in this process. For example, a surrogate model is a simplified representation of an experiment or computer simulation that would otherwise be difficult, time-consuming, or expensive to compute or observe. Creating a reliable surrogate model requires understanding of the domain-specific data it was created with. This especially impacts feature engineering during data analyses projects, e.g., surrogate modelling. Feature engineering is an umbrella term for several tasks, such as feature extraction, feature construction, and feature selection. These tasks highly depend on domain knowledge and greatly influence the prediction quality of the resulting models [WSR+20; HZC21].

Further, metadata is a key enabler for data analyses but at the same time often neglected in software applications, as described by Gray et al. [GLN+05]. Metadata are pieces of information that describe data and give structure and context to them. They can also describe the work activities that handle the data in virtual product development. In doing so, they support data discovery, exploration, and even analyses. Correspondingly, a suitable metadata model needs to describe not only data with metadata, but also the work activities of product development projects that produce and consume these data. This makes it possible to represent the dataflow in product development projects. It makes a metadata model familiar to domain experts, as this dataflow corresponds to the way they use software applications and work with their data [SP16]. Metadata about work activities also help domain experts to better compare and reproduce data, as well as to interpret and assess data quality more easily [DF08; HDB17].

Additionally, a metadata model that retains the relationships and dataflow between data and activities can then be used to discover process models for development projects [ZRS+22]. Thus, it makes process discovery applicable and tailored to projects in virtual product development. This is desirable

because organizations need knowledge about their real-world processes in order to improve and optimize them.

Literature comprises various generic metadata models for heterogeneous data formats [HSG+17; SSF+19; QHV16]. However, they do not address all challenges that are relevant to data in virtual product development. There also exist various domain-specific metadata models [VEC04; VFP+05; MSF+10; HSRW12]. Nonetheless, Willis, Greenberg, and White [WGW12] note that such an approach creates further separation between different disciplines of virtual product development and hinders interdisciplinary work.

1.2 Research Challenges

This thesis aims to advance the current state of digitalization in product development through metadata. For this reason, the thesis deals with three interrelated research challenges.

RC_1 Heterogeneity and Isolation of Data in Virtual Product Development

This challenge concerns the large amounts of data generated in projects of virtual product development. These data have different as well as changing formats and structures, which are often not known to the users [CSP13]. Suitable data and metadata management for the domain of virtual product development has to address these difficulties imposed by the heterogeneity of their data formats.

Additionally, data in virtual product development are often stored in separate data silos, each of which is managed by a specific organizational unit of a company [CKC20]. As stated by Che, Safran, and Peng [CSP13], as well as Gray et al. [GLN+05], such data silos limit the potential for gaining insights through data analysis, since each silo can only be analyzed in isolation. To identify relevant connections between data sets from multiple silos, domain experts need metadata that describes the data.

To address this challenge, a tailored metadata model for virtual product development needs to handle this heterogeneity of data formats and connect all data of the isolated silos.

RC_2 Difficulties Integrating Domain Experts into Data Analyses

This challenge is that data of virtual product development are often complex and domain-specific. That makes data analyses with these data difficult and domain experts are necessary to interpret the data as well as the results of data analyses. For this reason, data analyses in virtual product development benefit from domain experts who understand the physics behind their data [Via13]. Especially feature engineering highly depends on domain knowledge and greatly influences the prediction quality of trained models [WSR+20; HZC21]. Although some automation exists for feature engineering, the resulting models often no longer rely on physical foundations [TWP+19; WSR+20]. Algorithms usually do not understand any physical laws and, e.g., illegally calculate values even with incompatible units of measurement.

To address this challenge, domain experts can be incorporated into the process to ensure the physical correctness of trained models in data analysis projects.

RC_3 Difficulties Discovering Processes in Virtual Product Development

This challenge is that existing process discovery algorithms are not well suited for finding process models in virtual product development data. Data of projects in virtual product development are usually stored in dedicated information systems, e.g., PLM systems. Such systems and their data offer extensive provenance information [GT08; HDGL13]. However, process discovery algorithms expect to operate on event logs, i.e., a sequentially recorded stream of time-stamped activities [van+12]. Event logs are typically found in machines used in the manufacturing industry [BWD+21]. Converting PLM data to this event log format is possible, but the rich information about provenance and other metadata from PLM systems is then lost. Information about relationships between activities and the data they produce or consume can then not be utilized by process discovery algorithms.

Furthermore, the characteristics of projects in virtual product development are difficult to recognize for existing process discovery algorithms [dSMF+19]. While the basic building blocks of these projects are known in advance, the actual progressions of projects can be highly unstructured. Depending on the

specific product to be developed, different requirements, constraints, and teams of engineers lead to very different and dynamic projects. Here, existing process discovery algorithms often attempt to derive a single and generic process model for all variants of these unstructured development projects. This then results in a huge and convoluted process model which is hard to understand and where the individual project variants are lost and can, therefore, no longer be analyzed [Gv07].

To address this challenge, a novel approach to process discovery is needed that overcomes all these problems to make process discovery applicable and tailored to virtual product development.

1.3 Research Contributions of this Thesis

This section provides an overview of the research contributions of this thesis. To create contributions that can adequately solve the research challenges, the Design Science paradigm by Hevner et al. [HMPR04] was used. It provides a suitable basis for developing these contributions in a practical environment in close cooperation between the university and an industrial partner, here, MANN+HUMMEL GmbH. The core business of MANN+HUMMEL GmbH is filtration and their products cover a wide range of applications. The contributions were evaluated under real-world conditions at the industry partner to ensure their applicability in practice. Figure 1.1 gives an overview of the contributions and the corresponding chapters in this thesis. Preliminary versions of these contributions have been published in different conference proceedings [ZRKM20; ZRKM21; ZRS+22].

C_1 A Metadata Model to Connect Data in Virtual Product Development

This contribution addresses research challenge RC_1 and proposes a metadata model that offers a tailored approach to managing data and metadata in virtual product development. The metadata model provides the means to describe all data and work activities of product development projects in an interconnected structure. Work activities are all activities in product development projects that produce or consume data. They represent the actual work done either by an employee or a machine through computer software. The interconnected

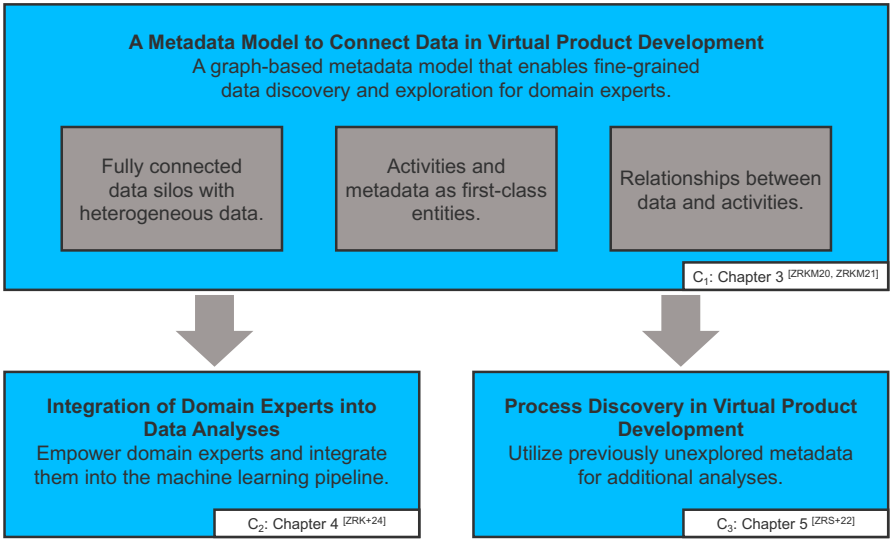

Figure 1.1: Blue boxes depict the contributions of this thesis and their corresponding chapters. Gray boxes depict the essential features of the foundational contribution C_1.

structure eliminates data silos and provides a connected view on all data, metadata, and activities across organizational units. The explicit representation of work activities and their relationships to data and metadata natively represents the dataflow of development projects as it is familiar to domain experts. The metadata model and architecture is implemented as a prototype to validate its feasibility by applying it to a real-world use case. This use case shows that the metadata model facilitates better access to data for domain experts and empowers them to discover, explore, and analyze their data.

C_2 Integration of Domain Experts into Data Analyses

This contribution addresses research challenge RC_2 and describes an approach for the creation of data-driven surrogate models with domain experts integrated into data analysis projects. In this real-world use case, domain experts are involved in the data-driven creation of surrogate models through participation during feature engineering. The approach integrates knowledge from

domain experts, enhancing the reliability and accuracy of the resulting models. Data-driven creation of surrogate models allows updating them when new or additional data becomes available, continually improving their quality. A case study demonstrates the application of this approach and evaluates its results. As a consequence, the physical correctness of the surrogate models is ensured, while still benefiting from an automatable, data-driven machine learning pipeline. This empowers domain experts and promotes the democratization of data science and analytics.

A flexible selection of data is essential for this process [ZRKM20]. Selecting suitable training data sets has a high impact on prediction quality. This requires domain experts who understand the physics behind the computer simulations that are to be approximated. Based on the metadata model from C_1, domain experts can apply their domain knowledge, and discover as well as select the most appropriate datasets for training a particular surrogate model [ZRKM21].

C_3 Process Discovery in Virtual Product Development

This contribution addresses research challenge RC_3 and proposes a novel approach to process discovery. It makes process discovery applicable and tailored to product development projects. The approach uses the metadata model from C_1 and its graph-based data structure that is able to represent all activities and data of product development projects. Graphs are capable of representing the provenance information available in PLM systems [ABD+14]. Further, the approach describes how this graph structure facilitates searching for frequent patterns. Patterns are frequently occurring sequences of activities that accurately describe a subset of a larger process. Instead of a single convoluted and approximated process model, patterns allow to describe each variant of unstructured processes as a set of accurate and precise patterns. The contribution discusses Frequent Subgraph Mining (FSM) algorithms to discover patterns in the graph structure representing product development projects and describes the application of these algorithms step by step. The approach is evaluated with a prototype and generated data that represents common product development projects. This shows that the approach can discover all relevant patterns in the data and how well FSM algorithms can handle errors and noise in the data.

1.4 Thesis Outline

The rest of this thesis is structured as follows:

Chapter 2 — Background: This chapter gives a more detailed overview of data in virtual product development and their characteristics. Further, it introduces the fundamental concept of a data lake and summarizes the topics metadata and metadata management.

Chapter 3 — A Metadata Model to Connect Data in Virtual Product Development: This chapter presents research contribution C_1 that solves research challenge RC_1 and forms the basis for the subsequent contributions. It first details specific challenges of a metadata model for virtual product development based on a use case and literature review. It then presents Graph'MeMo, a graph-based metadata model that solves these challenges and demonstrates it in a prototypical implementation and real-world case study. The chapter finishes with a discussion based on the introduced challenges and case study.

Chapter 4 — Integration of Domain Experts into Data-driven Surrogate Modeling: This chapter presents research contribution C_2 that solves research challenge RC_2. It describes an approach to better integrate domain experts into data analysis projects during feature engineering. The approach is explained in detail with step-by-step instructions. The application of this approach is then described with a real-world case study. The chapter finishes with an evaluation of the approach that investigates the improvement in prediction accuracy and reliability that could be achieved.

Chapter 5 — Graph-based Process Discovery in Virtual Product Development: This chapter presents research contribution C_3 that solves research challenge RC_3. It first investigates the mismatch between existing algorithms for process discovery and the data available in projects of virtual product development. Based on the metadata model Graph'MeMo, it then describes an alternative approach that searches for patterns in product development projects with FSM algorithms. After an algorithmic and step-by-step description of the approach, it is applied to a randomly generated data set that mimics the conditions of real projects in virtual product development and evaluates the findings.

Chapter 6 — Conclusion and Future Work: This chapter summarizes the presented contributions of this thesis and finishes with an outlook on future work.

Chapter 2

Background

This chapter introduces core concepts and background related to the contributions of this thesis. Section 2.1 presents more detailed information about the data typically found in virtual product development with a focus on computer-aided technologies. Section 2.2 describes concepts around the topic of big data storage with a focus on data lakes and metadata.

2.1 Data Characteristics in Virtual Product Development

A company with strong activities in virtual product development inevitably collects large amounts of heterogeneous data over time. The majority of these data is created by so-called *Computer-aided X* applications. This umbrella term combines Computer-aided Design (CAD), Computer-aided Engineering (CAE), and Computer-aided Testing (CAT), which are often used in conjunction with each other. To better understand the requirements and challenges imposed by these data, this section describes their characteristics as found at the cooperating industrial partner of this thesis, MANN+HUMMEL GmbH. An overview of typical data in this domain is summarized in Table 2.1.

Simulation applications typically perform Computational Fluid Dynamics (CFD) and Finite Element Analysis (FEA) calculations, which may require 2D or

3D CAD geometries as input data. These can be created with external computer applications or with a graphical editor sometimes integrated into the simulation application itself. Typical file formats for such geometries are binary files that have proprietary formats. In addition, there is also an ISO-standardized format for CAD called Jupiter Tessellation (JT)[1].

Different simulation applications also require different and often large input data, which can be used as a basis for calculations. These may be complex 3D geometries, often in application-specific and proprietary data formats. Some simulation applications create 3D geometries based on rich media files, such as images from an X-Ray Computed Tomography (XCT) scan. Parameters of a simulation application, such as the numerical method to be used for the calculations (e.g. CFD or FEA), are generally stored in application-specific formats. Due to small sizes of such parameters, simple text files in proprietary formats as well as documents based on Extensible Markup Language (XML) or JavaScript Object Notation (JSON) are common for storage.

Simulation applications usually generate output data in application-specific binary files or occasionally formatted as Comma-separated Values (CSV). These files are very large and further processing is often required to derive user-desired metrics from simulation results, such as product performance characteristics. Some simulation applications can also generate graphically enhanced output in the form of natural language text, images, or videos. These serve as visual reports and are usually not intended for the simulation engineer but for layman usage. The information in such a report is usually limited to simple key performance indicators and does not contain the information that a detailed analysis of the output binary files can provide.

CAT data formats depend on the underlying test benches they are produced with, which are purchased from various suppliers. Since different suppliers rely on their own data formats, CAT data is as diverse as data from simulation applications. Output data in proprietary binary and text files represent the raw and unfiltered output of test benches. This often involves a direct recording of measured values and sensor data over the duration of the test. In the case of, e.g., a stress test, the force applied to a test object over time is recorded along with the deformation caused to the test object. Such raw data is very

[1] ISO 14306:2017: https://www.iso.org/standard/62770.html

Table 2.1: Typical characteristics of CAE, CAD, and CAT data in virtual product development.

Purpose	Common Data Formats	Size
Simulation input data	3D CAD geometries; binary files in proprietary format; open-standard JT format	1 MB – 100 GB
Simulation parameters	Plain text files in proprietary format; XML documents; seldom binary files	1 KB – 1 MB
Simulation output (raw output data)	Binary files in proprietary format; CSV-formatted files	1 MB – 1 TB
Simulation output (exported data)	Rich text and media files; HTML documents	1 KB – 1 GB
Test bench results (raw output data)	Plain text and binary files in proprietary format	1 KB – 10 GB
Test bench results (exported data)	Plain text files in proprietary format; XML/JSON documents; CSV-formatted files	1 KB – 10 MB

specific with regard to the test bench. For example, recorded sensor values may not be stored in known physical units, but instead may be available as internally measured quantities, which must first be converted to known units. In addition, special export formats are commonly available, which serve as input for visual result representations. These formats reduce the file size and complexity to an information level that makes sense for the user, but they also do not contain all the information or the insights that are present in the raw data.

2.2 Data Lakes and Metadata

Data lakes are a modern approach to data storage and analysis that are becoming increasingly important in the realm of virtual product development. Their foundational concept lies in the centralized storage of heterogeneous data, from raw and processed to structured and unstructured, while maintaining their original form. This approach stands in contrast to traditional data

analytics platforms, such as data warehouses, where data must be cleaned, transformed, and structured before storage.

Several reasons make this approach suitable to virtual product development data. It enables all heterogeneous data formats to be brought from their isolated silos to a single conceptual location [JQ17]. The high scalability of data lakes enables the storage of increasingly growing amounts of data [Mat17]. Storing the raw data is especially useful for domain-specific data so that it can be useful for future use cases that require other types of data preparation [TWP+19]. Combined with an analytics platform, it is able to serve a large variety of users and use cases.

Metadata enriches these diverse data by providing crucial context. It informs about the source, structure, and content of the data. It serves as a directory, keeping the data organized and also acts as a key for efficient retrieval of specific data from a vast data lake. Metadata aids users in discovering relevant data, empowers them to interpret these data, and extract valuable insights.

The concept of the data lake and metadata thus forms an essential foundation on which this thesis is built. Therefore, the data lake is introduced in Section 2.2.1, followed by a brief overview of metadata and their role in a data lake in Section 2.2.2.

2.2.1 Data Lakes

A data lake is characterized as a centralized repository, designed specifically to accommodate heterogeneous raw data from diverse sources for analytics. Centralizing multiple data sources is not an entirely novel concept and already existed in so-called *dataspaces*. However, a dataspace does not materialize the data from its sources in a centralized location. Instead, data are logically connected and remain within their original source systems [HFM06; JQ17]. This configuration impacts possibilities for data integration and keeps oversight over data and data governance at the originating source system.

The initial definition of the term data lake traces back to a blog post by James Dixon from 2010 [Dix10]. Dixon's data lake is defined as a storage facility for raw data of various formats from a single source. The literature quickly transitioned from this definition to a broader version of a data lake that receives data from multiple sources [ML16]. Today, many definitions of

data lakes exist in literature and are sometimes ambiguous or contradict each other, as analyzed by Giebler et al. [GGH+19].

Fang [Fan15] and Mathis [Mat17] consider a data lake that stores structured, semi-structured, and unstructured data in their unprocessed raw format. Data may be retrieved from multiple sources, potentially in high volumes. For further use, such as data analyses, a data lake supplies data to other entities. Efforts regarding data integration are not made in advance but only when needed. Similarly, data is preprocessed and transformed not in advance but as part of a data processing pipeline for an actual use case.

Data lakes and data warehouses are often compared due to their shared principle of centralizing data from multiple sources [ML16; MT16]. Notable differences exist in their treatment of the data. A data warehouse pre-processes data according to the Extract-Transform-Load (ETL) principle, storing only transformed data generated in the transformation step. In contrast, a data lake stores raw data without pre-processing, implementing an Extract-Load-Transform (ELT) principle. Here, transformation occurs for each unique use case, while a raw and untouched version remains in the data lake. Given the assumption that each pre-processing step reduces information, this strategy aims to deliberately accommodate future and unknown use cases, particularly in the field of data analysis.

The type of data stored also distinguishes these two systems. A data warehouse predominantly stores structured data in relational data stores. In contrast, a data lake intentionally extends its storage concept to semi-structured and unstructured data [ML16; MT16].

Because data lakes store unprocessed data without a pre-existing schema, this raw data must be stored in its original form. This implies that a schema must be generated at the time of use for further analysis [Mat17].

Another advantage of data lakes is their ability to handle large volumes of data. As virtual product development becomes more complex and adopts high-fidelity simulations and intricate CAD models, the amount of data generated increases. Data lakes, often built on distributed infrastructure, are designed to handle such large-scale data, offering virtually unlimited storage capacity that is simple and cheap to extend [Mat17].

All these properties of a data lake provide a degree of flexibility that makes data lakes well suited to address the data-related difficulties faced in virtual product development. Product development in a digital environment generates data different from more traditional relational data, as described in Section 2.1. In a data lake, raw data from various stages of the product development process may be stored and cataloged without prior processing, retaining their original form. This capability enables organizations to preserve the entirety of the value of their data for future use.

2.2.2 Metadata and Data Management

Metadata give context to data [Ril17]. They cover a wide range of data that describe data. For example, a music album may have metadata such as an artist, a publisher, a file format, and a genre. Metadata may have different structure and formats and may be relevant to different user groups, depending on their context. For example, music listeners may be interested in the genre of an album for music discovery, while a music player may utilize the file format information to facilitate playback. Different approaches and standards to categorize metadata exist [GHP10; Ril17; RZ19]. For example, metadata may be grouped and classified either by their function or structure [SD21].

Data management in a data lake refers to the process of organizing, classifying, and maintaining the data stored in the data lake. Metadata takes on a significant role in the context of a data lake and its data management. Given that a data lake contains data of varying degrees of structure and often no schemas, it may turn into a so-called *data swamp* without appropriate metadata and data management [HGQ16]. A data swamp is an unmanaged and disorganized version of a data lake and leads to inaccessibility of data, poor data quality, inefficiencies, and low usability. Such a degenerated data lake cannot deliver the anticipated benefits of data lakes and undermines its purpose. Thus, it is essential to prevent this scenario through diligent metadata and data management.

A metadata model plays an important role in enabling effective metadata management and data management in a data lake [SD21]. It provides a framework for organizing and categorizing metadata, defining attributes, and implementing technologies to optimize data access and analysis. By providing

detailed information about the structure, format, source, and other attributes of data, a metadata model improves the usability and accessibility of the data. A metadata model and its metadata provide valuable context and information about the data stored in the data lake, thus preventing a data swamp. This allows users to easily find and understand the data they need for various analyses and decision-making processes.

Chapter 3

A Metadata Model to Connect Data in Virtual Product Development

This chapter proposes a metadata model that offers a tailored approach to managing data and metadata in virtual product development. The metadata model provides the means to describe all data and work activities of product development projects in an interconnected structure. This eliminates data silos and provides a connected view on all data, metadata, and activities across organizational units. The explicit representation of work activities and their relationships to data and metadata natively represents the dataflow of development projects as it is familiar to domain experts. The metadata model is implemented as a prototype to validate its feasibility by applying it to a real-world use case. This shows that through the use of this metadata model, the prototype facilitates better access to data for domain experts. They can use it for data discovery, data exploration, and data analyses to optimize their products.

This chapter presents contribution C_1 and addresses research challenge RC_1. The contents of this chapter are an extended and revised version of two earlier publications by the author [ZRKM20; ZRKM21].

The remainder of this chapter is structured as follows: A use case and derived challenges for a metadata model in virtual product development are presented in Section 3.1. Section 3.2 discusses related work. The conceptual metadata model and a graph-based logical realization is introduced in Section 3.3. Section 3.4 describes the prototype, the case study, and the results of the validation. Section 3.5 concludes with a summary of this chapter.

3.1 Challenges for a Metadata Model in Virtual Product Development

A partnership with MANN+HUMMEL GmbH helped identify relevant challenges for a metadata model in virtual product development from an industry perspective. These challenges have further been aligned and backed up by a literature review that provided an academic perspective. See Section 2.1 for a characterization of data in virtual product development.

Section 3.1.1 outlines a generic workflow in virtual product development projects. Section 3.1.2 presents a real-world use case that may benefit from digitized product development. Section 3.1.3 finishes with domain-specific challenges for a metadata model in virtual product development.

3.1.1 Product Development Project Workflow

Together with engineers from MANN+HUMMEL GmbH, a generic workflow for product development projects was identified. It combines various internal data sets from different organizational units of the company. This workflow also aligns with a more universal definition of virtual product development [HDGL13]. Figure 3.1 represents the workflow with its individual steps, numbered from 1 to 9. The blue boxes represent data sets, while the yellow boxes with a dotted line represent work activities, i.e., steps performed by employees in the project. The arrows connect boxes and represent the dataflow in

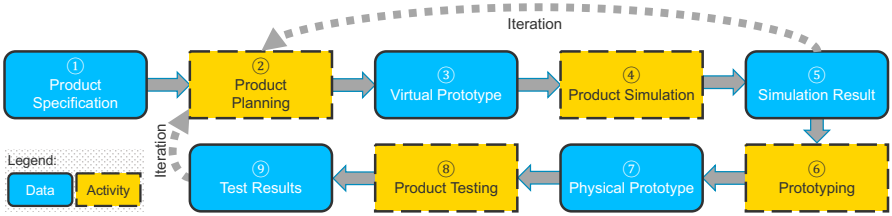

Figure 3.1: Illustration of a simplified product development project involving computer-aided product simulations (steps 3 to 5) and product testing (6 to 9) [ZRKM21].

product development projects. They show which of the data sets are consumed or produced by which activities.

The workflow starts with a product specification (step 1) that may be provided by an external customer or by an internal development unit. Subsequently, it is used by domain experts to plan the product and to design a virtual prototype (steps 2–3). In the next activity, this virtual prototype is simulated with a Computer-aided Engineering (CAE) application (step 4). The resulting simulation data (step 5) are then used to draft a design plan for a physical prototype (steps 6–7), which is tested with a Computer-aided Testing (CAT) application (steps 8–9). If the original specification is met, the newly developed product can be transferred to production planning. To reach this point, several iterations of repeated product planning, simulation, and prototyping steps may be necessary.

This workflow illustrates the various disciplines in a product development project that generate large amounts of heterogeneous data, i.e., product design (CAD), product simulation (CAE), and product testing (CAT) (see Section 2.1). The strong data heterogeneity is further increased by a wide range of competing computer applications, which rarely meet open standards with regard to data. Simulation applications typically require large 3D CAD geometries as input, which are usually stored in proprietary file formats. Project-related simulation parameters, such as the numerical method to be used for the calculations, are generally stored in smaller application-specific formats. Their output data are large application-specific binary files that need further processing to derive user-desired metrics such as product characteristics. The data formats of CAT

data depend on the test benches used. Different suppliers rely on their own data formats, which are usually binary or text-based file formats.

3.1.2 Use Case: Feedback in Virtual Product Development

Data from the disciplines Computer-aided Design (CAD), CAE, and CAT are often stored in separate data silos, each of which is managed by a specific organizational unit of a company. As noted by Che, Safran, and Peng [CSP13], such data silos limit the potential for gaining insights through data analysis, since each silo can only be analyzed in isolation. To identify relevant connections between data sets from multiple silos, domain experts need context that describes the data. Context may be provided by metadata that describe properties of data, such as the dimensionality of a CAD geometry. In addition, information about the origin of a data set can describe which computer applications and which parameters were used for data creation, e.g., the name of a simulation software or the numerical calculation models used. Furthermore, issues of data quality must be identified and solved before subsets of the data are used for analysis. Metadata can aid this by providing domain experts with more context to better assess outliers and inconsistencies in their data.

Related to the workflow shown in Figure 3.1, connecting data from the two silos product testing (CAT) and simulation (CAE) can reduce the number of necessary iterations and, thus, reduce the duration and costs of product development. If simulation or test results (steps 5 or 9) indicate that the product specification cannot be met, it is necessary to start a new iteration with the planning step 2. The goal is to make the decision to start a new iteration as early as possible, i.e., based on the simulation results in step 5. This can avoid expensive and time-consuming product tests, i.e., all steps from 6 to 9. However, this is only possible if the simulation models accurately simulate the real physical tests.

Even with the current state of the art in computer simulation, physical correctness is not a given: The accuracy and reliability of results vary greatly depending on the specific problem area [Cou+12]. With data and metadata management that offers a connected view on data silos of CAE and CAT, data may help to optimize simulation models. The numerous data from historical

physical product tests (step 9) may be used to increase the accuracy of the simulation results (step 5) in a feedback loop.

To realize this, a connection of CAE and CAT data is necessary. It needs to be possible to connect simulation data, simulation models, and corresponding data from product tests on various levels of granularity. More precisely, analyses on historical testing data may be used to improve corresponding simulation models. This can be achieved, e.g., by calibrating predefined constants in the simulation models, which are obtained through Gaussian Process regression on historical testing data. This calibration has to be done for various parameters and the problem area itself: materials, product dimensions, intended purpose, and many more. All permutations require their own specifically optimized calibration constants. Consequently, no predefined selection of CAE and CAT data can be used. Different calibrations are necessary for each permutation of parameters and problem area. A data set for such a regression analysis must be selected and connected dynamically and flexibly, with the goal of creating a likewise automated feedback loop. The results are highly accurate simulation models for specific use cases.

In such a feedback loop, each additional CAT data set may improve the simulation models. Automatically creating connections between corresponding data sets from multiple silos, dynamic and flexible data selections as well as data analyses are all highly dependent on available metadata. In addition, knowledge about the reliability of the simulation results and their deviation is also gained, which supports decision-making in the course of a product development project, as engineers can better assess the available results. This enables domain experts to decide more reliably and, thus, more frequently at an earlier stage (after step 5) whether a product requires an iteration back to the planning step 2.

3.1.3 Challenges for a Data and Metadata Model

Within research challenge RC_1, this section presents three challenges MC_1, MC_2, MC_3, which need to be addressed by a metadata model for virtual product development. Aside from these challenges, there are also desirable properties for metadata models in general, which will be briefly mentioned first.

One important property is the **flexibility** of a metadata model. It must be able to handle intermittent changes in data schemas and formats in the source systems. The available metadata might change over time and new information can be available in the future while other information might be absent. For example, when a computer application used in product development is updated, new metadata may become available or previously available metadata may disappear. Another important aspect is the metadata model's **generality**. In order to connect data silos across organizational units in a company, different use cases and their various data sets must be supported. One more general property is **simplicity** regarding the structure and handling of the metadata model. The metadata model and the underlying data lake should be used directly by the domain experts themselves for the extraction and selection of data and metadata. Accordingly, the metadata model must not require any significant database knowledge, so that the domain experts can understand and use it with the least possible training required.

The following describes the specific challenges a metadata model for virtual product development has to address.

MC_1 **Isolated data silos with heterogeneous data sets**: A metadata model must be able to provide a connected view of all data of product development projects across the boundaries of various organizational units. Such a connected view allows domain experts to access data sets that were previously stored in data silos to which they had no access. Data in virtual product development exist in different formats and structures, which are often not known to the users [CSP13]. Hence, a level of abstraction is required that allows metadata to be linked to all available heterogeneous data. This facilitates intuitive data exploration tools tailored to domain experts. It allows them to explore the connected data sets and select subsets they need for further projects, e.g., for data analyses. In case of data analysis projects, they can also benefit from the connected view of all data [CSP13; GLN+05].

MC_2 **Activities have to be represented as first-class entities with their own metadata**: Activities, as understood here, are all work activities in product development projects that produce or consume data. They represent the actual work done either by an employee or a machine. In both cases, there is software involved. Employees may process data either indirectly, e.g., by

running a computer simulation, which generates data, or directly by manually processing data using spreadsheets or similar tools. Machines in testing run benchmarks controlled by CAT software that generates test reports.

For a metadata model aimed at domain experts in virtual product development, it is important to not only represent data itself, but also activities they were generated with in their role as first-class entities. Capturing the activities with their metadata improves comparability and reproducibility of data sets, since the activities that produced them can be analyzed explicitly [DF08; HDB17]. For example, metadata of a simulation activity may include the numerical calculation method used for the calculations. This has several implications on the data set it produced, e.g., the uncertainty or accuracy associated with the simulated results. In general, metadata of an activity influence the way a data set is interpreted and what can be done with it, e.g., if it is fit for subsequent data analyses. Therefore, activities are a core element of a product development project and domain experts consider them as important as the data sets themselves. Hence, these activities have also to be associated with numerous metadata that are relevant for subsequent processing and interpretation of data sets.

MC_3 **Relationships between data and activities have to be represented**: The representation of relationships in a metadata model makes it possible to directly link all data with the activities that produce or consume these data. This allows accurately modeling the dataflow of product development projects, e.g., in a similar way as the dataflow shown in Figure 3.1. Such a dataflow represents which data are related to which activities and, thus, depicts the project progression in a way as it is familiar to domain experts working on it [SP16]. Consequently, they can intuitively navigate through such a metadata model, explore it and easily find desired data. Explicit descriptions of relationships between data, activities, and metadata allow domain experts to see the data lineage, i.e., how data were created and how they were processed over time. This further simplifies the interpretation of the data by domain experts [Bos02; GLN+05]. For example, they can more easily rate the quality of their data and assess their applicability to other use cases.

3.2 Related Work

This section categorizes related work into three groups: metadata models, workflow modeling techniques, and commercial computer applications. The metadata models are further divided into generic models, which attempt to support metadata of any domain, and specialized models, which are tailored to specific use cases in simulation and scientific data management. A summary is compiled in Table 3.1.

Generic Metadata Models Sawadogo et al. [SSF+19] propose Metadata model for Data Lakes (MEDAL) based on a hypergraph, i.e., consisting of hypernodes and hyperedges. Each hypernode represents a data object and contains a simple graph, whose nodes represent different versions and representations of this data object. Each node in this simple graph is associated with its own set of attributes describing a specific version of the data. Attributable edges between these simple nodes represent modifications that lead to new versions. Hyperedges connect multiple hypernodes to group and aggregate data objects. This creates three categories of data aggregation: grouping based on similar attributes, grouping based on similarity metrics, and parenthood relationships. These fixed aggregations are useful for accessing data. While groupings based on attributes and metrics facilitate the search for similar data, parenthood grouping enables hierarchical access patterns.

Hellerstein et al. [HSG+17] developed a metadata model called *Ground* that also associates each data object with its own graph. Instead of a hypergraph, Ground uses three distinct graphs that are interconnected: version graphs track versions of data objects, model graphs describe dependencies between data objects, and lineage graphs attribute provenance information. A node of a *version graph* represents a particular version of the associated data object. Edges connect two version nodes of the same data object and represent the transition from one version to the other. Ground adds separate *model graphs* to describe dependencies between data objects. Edges between nodes of a model graph represent abstract dependencies between different data objects. Specific semantics of edges in a model graph are to be defined by client applications themselves, as appropriate for their application logic. Each node of a model graph always refers to a specific version of a data object, i.e., a node of a model

graph is linked to a node of a version graph. In a third type of graph, the *lineage graph*, versions of data objects or a version transition can be annotated with usage information. Nodes or edges in a version graph are connected to a node in the lineage graph, which is either a so-called principal or workflow node. A principal is defined as a manual modification through a user and a workflow is an invokable specification of code.

The Generic and Extensible Metadata Management System (GEMMS) proposed by Quix, Hai, and Vatov [QHV16] is specifically designed to ease data integration in data lakes. A central part of this system is an automated extraction of metadata from files. Metadata are categorized as structural or semantic and annotate abstractions of files or subsets of files from which the metadata was previously extracted, called units. Through this abstraction, GEMMS can describe data at different levels of granularity. It further categorizes the metadata to make them queryable and explorable. Annotations with structural metadata additionally aim to recreate the schema of the data described. However, this relies on the availability of parsers for the file formats whose metadata are to be extracted.

All three metadata models provide abstractions to attribute heterogeneous data with metadata (MC_1). However, they all lack the ability to represent work activities of virtual product development as first-class entities and to explicitly associate them with metadata (MC_2). Likewise, they cannot describe the relationships between data and activities in a way that represents the dataflow in product development projects (MC_3). Ground can link version transitions with manual user interactions or code executions. However, corresponding principal or workflow nodes are only attributed to individual data objects and cannot explicitly represent a dataflow. This is because it is not possible to define links describing input-output relationships between several processes. GEMMS cannot represent work activities and relationships between data at all. Both graph-based approaches (MEDAL, Ground) can attribute relationships among data objects with metadata, but these relationships do not represent a dataflow between work activities. MEDAL can describe transitions from one version to another with relationships, but it does not allow for explicitly describing the activities carrying out these version transitions.

Table 3.1: Summary of related work. ● = fulfilled; ◑ = partially fulfilled; ○ = not fulfilled [ZRKM21].

	MC_1	MC_2	MC_3
Generic Metadata Models [HSG+17; QHV16; SSF+19]	●	○	○
Specialized Metadata Models [MSF+10; VFP+05; VEC04]	◑	○	○
Workflow Modeling Techniques For example, BPMN [Obj14]	◑	◑	●
Commercial Computer Applications	◑	○	○

Specialized Metadata Models Several proposals for data and metadata models exist within the scope of simulation or scientific data management. Vasilakis, El-Darzi, and Chountas [VEC04] focus on specific event simulations and suggest to store simulation output data in a data warehouse. Due to their scope of specific event simulations, a data schema can be defined in advance. The data warehouse technology allows them to leverage existing Online Analytical Processing (OLAP) tooling to perform predefined data analyses. The approach proposed by Valle et al. [VFP+05] is tailored to the specific problem of Computational Fluid Dynamics (CFD) simulations. Their Scientific Data Bag (SDB) is a container format that stores semantic information as metadata to describe heterogeneous files. The SDB enables applications to implement visualizations of simulation results. For the use case of spatial event simulations, Hoppen et al. [HSRW12] present an object-oriented approach to data management. They create a central database as backend with a predefined data schema that is optimized for the spatial dimensions of their data. Matthews et al. [MSF+10] introduce the Core Scientific Metadata Model (CSMM) aimed at representing data generated by scientific studies as well as the metadata of these studies. The main design goal of the authors is the ability to extend metadata structures to generalize well for various types of scientific studies.

All three proposed solutions are tailored to specific use cases or problem domains. Each proposal requires an explicit a priori step of data modeling to create a data schema, which is very time-consuming for the heterogeneous

data in virtual product development (MC_1). It also needs to be repeated whenever the structure or availability of data or metadata changes. They are limited to their respective data silo and do not allow connecting data across boundaries of different silos and organizational units (MC_1). Furthermore, the proposals have a strict focus on describing data, but they do not allow to explicitly represent work activities (MC_2) and, thus, lack relationships between data and these activities (MC_3).

Workflow Modeling Techniques Approaches from a different angle are techniques to represent workflows or business processes. The Business Process Model and Notation (BPMN) is the major representative of such techniques [Obj14]. BPMN represents process steps as so-called *tasks*, which may be used to model the work activities and the dataflow between them as in product development projects (MC_3). Furthermore, BPMN allows associating these tasks with abstract representations of a data set. Thereby, it is possible to assign types to data sets and distinguish between input and output data of tasks. However, it is not possible to attribute data or tasks with arbitrary metadata (MC_1, MC_2). While BPMN annotations could be re-purposed for metadata, this has several drawbacks. BPMN has no metadata definitions in its core. This requires domain experts to spend much effort to decide on structures themselves. Moreover, it lacks the flexibility to express more than simple one-to-one relations.

BPMN and similar contenders were designed as visual language to specify and formalize business processes. This usually happens before processes are executed and once specified they are then executed many times. However, the use case for a metadata model as needed here is different. Data and metadata in a product development project are accrued while the project is executed or even afterwards. At that time, the metadata model is used to document an individual development project. The individual description of this project is likely not to be shared by any other project in the future and its documentation will not be used as an execution plan for other projects. The widespread use of BPMN provides a broad arsenal of tools that enable non-technical domain experts to work with BPMN. However, BPMN lacks readily available software to query a database of BPMN processes, although there exists academic research in this direction [Awa07].

Commercial Computer Applications Management of CAE data is a large field where various commercial off-the-shelf computer applications are available, e.g., Dassault Systèmes SIMULIA[1] or T-Systems Medina/SDM[2]. These offerings are familiar to domain experts due to their presence in industry. However, they are tightly coupled with dedicated simulation software for CAE applications and have an explicit focus on simulation software. Hence, they are less suited to create a connected view that includes data sets from other areas, such as CAT (MC_1). They also lack the capability to describe arbitrary work activities (MC_2) or relationships between data and activities (MC_3).

3.3 Graph'MeMo: A Graph-based Metadata Model

Due to the high heterogeneity of data in virtual product development, the metadata model proposed in this section separates storage of data from metadata. Data files are stored in their raw formats in a data lake which acts as storage backend and analytics platform [Fan15]. Graph'MeMo, the **Graph**-based **Me**tadata **Mo**del, serves as a way to index files in this storage backend.

Several reasons make a data lake suitable to virtual product development data. It enables all heterogeneous data formats to be brought from their isolated silos to a single conceptual location [JQ17]. The high scalability of data lakes enables the storage of increasingly growing amounts of data [Mat17]. Storing the raw data is especially useful for domain-specific data so that it can be useful for future use cases that require other types of data preparation [TWP+19].

The following Section 3.3.1 introduces the conceptual metadata model. Section 3.3.2 describes how to realize it with a graph-based logical model.

3.3.1 Conceptual Metadata Model

Figure 3.2 depicts the conceptual metadata model as Unified Modeling Language (UML) class diagram. In its core, the model consists of three main elements: data containers, activities, and contextual metadata. In the following, this section details on these elements and discusses how the metadata model addresses the challenges introduced in Section 3.1.3.

[1] SIMULIA: https://www.3ds.com/products-services/simulia/products
[2] Medina/SDM: https://plm.t-systems-service.com/en/medina-sdm

A *data container* is an abstraction of an identifiable collection of data located in a storage backend, previously defined by Reimann et al.; Reimann, Schwarz, and Mitschang [RRS+11; RSM14]. This level of abstraction allows the metadata model to represent data files independently of their format or structure. This way, it is possible to attribute heterogeneous data originating from any data silo involved in product development projects (MC_1). Each data container is composed with exactly one Uniform Resource Identifier (URI) that points to its storage location, e.g., to a file in the underlying data lake.

An *activity* element is an abstraction of a work activity that consumes and/or produces data in a product development project. Such an activity may be the creation of a product design in a CAD application or the calculation of a computer simulation. In the same way as with data containers, this level of abstraction for activities allows the model to conceptually represent all activities in product development projects as first-class entities in a metadata model (MC_2).

Input and output relationships associate an activity with the data containers it consumes as input and/or produces as output. These associations between data containers and activities describe the dataflow of a product development project (MC_3). They have many-to-many cardinalities to reflect all kinds of activities occurring in practice. Many activities both consume and produce at least one data container, e.g., a computer simulation that takes a product design to simulate the properties of this product. Nevertheless, some activities produce several data containers, but do not consume any, or vice versa.

Neither data containers nor activities directly contain a form of metadata. Instead, they may be associated with an arbitrary number of *contextual metadata* elements. This indirection makes it possible to attribute data and activities with any metadata regardless of their heterogeneity (MC_1, MC_2). A contextual metadata element must have at least one association with either a data container or an activity, specified by the logical $\{or\}$ constraint, as defined in the Object Constraint Language (OCL)[3]. A metadata element must not stand alone, because it is used to describe a data container or an activity. However, a single metadata element may describe multiple data containers or activities. Each metadata element is composed of at least one *attribute* that represents

[3]OCL 2.4: https://www.omg.org/spec/OCL/2.4/PDF

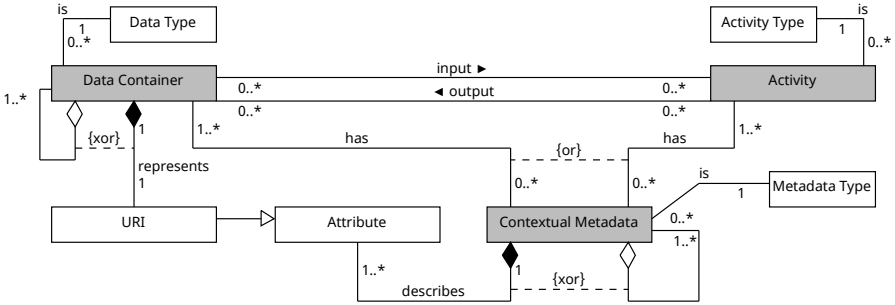

Figure 3.2: Conceptual metadata model as UML class diagram. The core elements are shown as gray boxes. White boxes show generalizations or composites of core elements [ZRKM21].

descriptive information for the associated data containers or activities. Thus, these attributes of contextual metadata elements describe their associated data containers or activities. An example attribute is the name of an employee who performed an associated activity. The URI of a data container is modeled as a specialization of such an attribute to ensure that any data container has such an identifier as metadata. It serves as an identifier and may point to a location in a storage backend.

The composition of attributes to metadata elements allows grouping multiple related attributes in one metadata element. This provides flexibility to the metadata model (MC_1). In case of a product design, a contextual metadata element may contain multiple attributes that characterize one of the materials the product is constructed of. Due to version changes of the utilized CAD application, the number of available attributes for material characterization may increase or decrease over time. In such a case, no major adjustments to the model are necessary. A new data entry will either omit a now missing attribute or add a new attribute while the main structure — the metadata element itself and its associations to data or activities — may remain unchanged.

Graph-based Metadata Model (Graph'MeMo) contains special forms of aggregation for data containers and contextual metadata. This aggregation allows a data container or metadata element to represent a collection of other data containers or metadata elements, respectively. This may be used to create virtual abstractions of multiple data containers. Domain experts can use this

aggregation to create hierarchical data structures comparable to those they are working with, e.g., an abstract data container may represent a folder of multiple files on their local disk. Note that such a container of containers must not have a URI, as it does not represent physical data. Hence, a logical $\{xor\}$ constraint prohibits that a data container may both aggregate other containers and be composed with a URI. Aggregation of contextual metadata further increases flexibility and enables hierarchical grouping based on, e.g., the topic of their attributes. In case one contextual metadata element contains attributes for time (e.g., hours, minutes) and another one for a date (e.g., year, month, day), both may be aggregated to have all timestamp-related metadata in one place. In a similar way to data containers, an aggregating contextual metadata element must not have attributes of its own.

The three core elements (data containers, activities, contextual metadata) are associated with exactly one individual *type*. This type allows domain experts to add domain-specific semantics to this otherwise abstract metadata model. For instance, it may be used to model the various computer applications used in CAE as types of activities. Thereby, a type may identify whether a simulation activity is implemented by OpenFOAM[4] or SU2[5], which are two popular open source software suits for computer simulations. In a conceptual model, this type element may refer to parts of a domain-specific concept model, a business glossary, or more expressive ontologies, e.g., the *Product Design Ontology* by Catalano et al. [CCF+09]. This ontology is designed for CAD data and CAE input data. It defines the types of data files that exist, what metadata they may have, and establishes a common vocabulary that improves interdisciplinary work between different areas of virtual product development.

3.3.2 Graph-based Logical Model

An appropriate choice for the logical level of Graph'MeMo is a graph structure. As summarized by Angles and Gutierrez [AG08], graphs have multiple advantages compared to other database models, such as relational, network and object-oriented models. For instance, graphs provide a simple structure that allows domain experts to intuitively model and access data. Furthermore, they

[4] OpenFOAM: https://www.openfoam.com/
[5] SU2: https://su2code.github.io/

Table 3.2: Mapping of conceptual metadata model to a property graph data model.

Conceptual Element	**node**	**edge**	**label**	**property**
Data Container, Activity, Contextual Metadata	✗			
Types of Data/Activity/Metadata			✗	
Attribute/URI				✗
input/output and *has* associations		directed		
Aggregation		directed		

natively support relationships between data and enable users to discover data with certain kinds of relationships. However, instead of a simple graph consisting of nodes connected by directed or undirected edges, a *property graph* data model is a more powerful structure [RN10]. A property graph is a directed multi-graph that can store values within its nodes and edges as properties and labels. This maps well to some concepts of Graph'MeMo's conceptual model.

Table 3.2 summarizes how individual elements of Graph'MeMo's conceptual model map to elements of a property graph in the logical model. Instances of the core elements (data containers, activities, contextual metadata) are each represented by a node in the graph. Each node has a label that corresponds to the conceptual type of the relevant data, activity, or metadata element. An attribute of a metadata element is represented by a property of the corresponding node in the graph. In addition, data container nodes get a dedicated property for their URI attribute. Note that activity and data nodes carry no other properties themselves, as they are not directly associated with attributes. Instead, activity and data container nodes are described by associated contextual metadata nodes that carry a property for each of their attributes. This provides an advanced position for metadata, e.g., for data discovery. A user may search for all data container nodes associated with a specific contextual metadata node, regardless of their type.

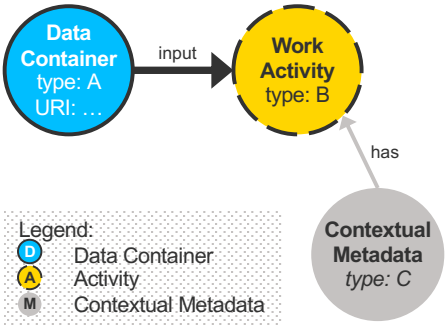

Figure 3.3: Illustration of a property graph structure [ZRKM21].

Directed edges between data container and activity nodes represent their *input/output* relationships (MC_3). The *has* associations that connect contextual metadata nodes to data container or activity nodes are represented via directed edges as well. An aggregation of data container nodes or metadata nodes is realized with a directed edge pointing from the aggregating node to the aggregated nodes. These aggregating nodes then must not carry any properties. The *is*, *represents*, and *describes* associations in the conceptual model are all part of nodes as their labels or properties. Hence, they do not need an explicit mapping.

Figure 3.3 shows an example of a graph structure realizing the conceptual metadata model. It shows a data container node and an activity node. The data container node represents a data set that exists in the underlying data lake (MC_1). It contains a URI as property that identifies the location of the represented data set in the underlying data lake. The data container node is connected to an activity node, which represents a work activity carried out by an employee or computer software (MC_2). The directed edge connecting data container to activity indicates the direction of the dataflow (MC_3). A contextual metadata node is connected to the activity node it describes. All nodes have a label that indicates their type in the conceptual model. For clarity, attributes are not depicted, with the exception of the URI. An exemplary instantiation of the graph structure realizing the conceptual metadata model for the workflow of a product development project is given in Section 3.4.3.

The resulting graph structure contains all data, activities, and their metadata in one connected view, across silos of organizational units. Domain experts are familiar with such a structure because the dataflow it represents corresponds to the way domain experts work in their product development projects. The following section describes a prototypical implementation and a case study that illustrates how this graph structure is created and used in practice.

3.4 Implementation and Case Study

This section describes the implementation of Graph'MeMo and the use case presented in Section 3.1.2. The implementation utilizes Graph'MeMo to connect data of virtual product development in a data lake. Section 3.4.1 introduces and uses Quality Function Deployment (QFD) to derive the architectural basis for the implementation. Section 3.4.2 presents a software architecture and implementation that embeds Graph'MeMo as central component to manage metadata. The use case is realized in Section 3.4.3, followed by a discussion in Section 3.4.4.

3.4.1 System Architecture

This section describes QFD to derive an overall system architecture that incorporates Graph'MeMo into a system to manage and provide data. QFD was first formulated in its original sense for quality assurance in 1966 [HSM97]. In addition, QFD is also actively applied in product development and is concerned with the task of placing the customer and their needs, also called *voice of the customer*, at the center. Applied to the process of product development, QFD focuses on the customer needs and their requirements, and, thus, increases product quality in the sense of customer satisfaction [HST15]. This is opposed to the traditional strategy of quality assurance that aims to increase product quality by reducing product defects. QFD does not aim to completely replace existing procedures in product development with the voice of the customer. Instead, QFD tries to include as many aspects and voices of others as possible, e.g., all departments of a company that are involved in a project. However, all these aspects must be focused for the sake of quality, which is achieved through

prioritization. Here, it is relevant which priority the respective aspects have in order to fulfill the customer needs.

The QFD process is versatile, as the core methodology of prioritization can be applied to many problems, including software products [HST15]. However, software development and the life cycle of software products has no manufacturing, as it exists in physical product development. Therefore, software QFD focuses on other phases of software development, such as the software development process itself. For example, on requirement engineering, which covers elements such as the specification, the analysis, and the validation of requirements as well as requirement management [Som10]. A generic form of QFD, such as the framework of QFD-ID, may be applied directly to software products for this purpose [HST15].

Beyond the software development process, there are further challenges for software developers where QFD can support them with its techniques [HPST12]. QFD frameworks and methods can be used to refine the requirements of a software product, but to also steer the system architecture of the software product to be developed. The system architecture has a direct influence on the characteristics of the finished software product, e.g., the expandability of the functionality or expected costs associated with long-term maintenance and servicing of the software. As such, those factors should be considered in the planning process of a system architecture [BCK03]. QFD can help to use given requirements in the planning process and combine them with such factors to optimize the system architecture and influence choices in the planning process through focussing and prioritization [AW16; HST15]. The use of QFD here can not only avoid common mistakes, but also provide documented traceability of the planning process.

3.4.1.1 QFD for Software

In contrast to classic QFD for physical products, there are no physical quality characteristics in the case of software products. A software product is evaluated based on its functionality and the provided services offered by the software product [Zul95]. Accordingly, QFD for software often takes a modified form. Usually, the House of Quality (HoQ) correlates customer needs with the quality characteristics of a physical product. However, in software, these quality char-

acteristics tend to be non-functional requirements. Therefore, Software QFD often correlates the customer requirements with the functional requirements of the software before proceeding with a HoQ.

One way to dovetail customer requirements with the design of an architecture is the so-called *Twin Peaks* model [Nus01]. Twin Peaks starts with high-level requirements and architecture, which it then increasingly sharpens in an alternating and iterative process. The requirements and architecture are at the center of this process and represent the two peaks the model is named after. Initially, Twin Peaks starts at a high level with generic requirements and a matching generic architecture. It then makes both aspects increasingly more specific by alternating iteratively between them. Both aspects influence each other: the increasingly more specific requirements influence the architecture, making it more specific in the process, and also vice versa. In particular, the back direction is to be emphasized here: the architecture can have a restricting effect on the requirements.

This process was implemented with QFD by Anang and Watanabe [AW16] as a method to realize the mutual influence of both aspects on each other. It is based on layered architectures and uses QFD to distribute customer requirements and system components on appropriate layers. A layered architecture consists of a set of n sequentially ordered layers. Components that reside in a layer m depend on components that reside in layers $p < m$, i.e., in the layers below m. However, they do not depend on layers above their own and those can be taken away without impeding functionality for the components. In a so-called strict layer architecture, components may only depend on the adjacent layer below their own, i.e., layer $m - 1$. The method of Anang and Watanabe [AW16] starts with an initial draft of a layer-based architecture and then refines and details it through alternating iterations between architecture and customer requirements.

This refinement exploits the diverse evolution of different components throughout the lifetime of the software product. While some components may change frequently during the lifetime of the software product to accommodate future requirements, some fundamental components are very unlikely to ever change. For example, a system for data provision must always offer the possibility to extract data from the system. However, whether such a system

must also provide the ability to analyze data is not a fundamental requirement and represents a customer request that may potentially develop in the future. This assumption can now be used to implement such requirements through layers of the architecture that provide or limit the necessary flexibility. For example, requirements that are unlikely to change may be implemented on a layer with little flexibility. In a layered architecture, these are the deeper layers that can access none of the layers above them. On the other hand, many upper layers may depend on such a deep layer, and, thus, it is crucial that deeper layers change as little as possible.

Anang and Watanabe [AW16] use a two-dimensional table to map functional requirements to architectural layers. The table's rows contain the requirements and the columns contain an initial draft of the layers. A user determines which requirement correlates how strongly with the envisioned functionality of a layer and enters this value into each corresponding cell of the table. Each correlation value is combined with an assessment regarding the assumed probability of a future change of the respective requirement. The result is an assignment of functional requirements to the layers on which they should be implemented.

3.4.1.2 System Architecture with QFD

The method proposed in Anang and Watanabe [AW16] can be applied here to influence and derive architectural decisions for a data platform implementing the metadata model Graph'MeMo. It starts with deriving initially unspecific requirements from the challenges previously introduced in Section 3.1.3. A requirement derived from challenge (MC_1) is the need to be able to ingest and store data sets and metadata from heterogeneous sources. Challenge (MC_2) implies that work activities of employees need to be stored, without specifying how this is to be done in detail. Challenge (MC_3) requires representation of relationships between the aforementioned data, metadata, and work activities. Additionally, the first and last challenges also reveal the need for this information to be provided to domain experts. This represents another overarching fourth requirement. All initial requirements are summarized below.

R_1 Ingest data from heterogeneous sources.

R_2 Storage of heterogeneous data and metadata.

R_3 Representation of work activities and relationships between data, metadata, and work activities.

R_4 Provide domain experts with applications and results based on the data, metadata, and work activities.

As for an initial architecture, the use case described in Section 3.1.2 is the primary goal for the system to be devised here. A central component to such a system is a software solution that utilizes Graph'MeMo to offer Create, Read, Update, and Delete (CRUD) [Mar83] functionality. For a system architecture with CRUD functionality at its core, a three-layer approach to the architecture is a suitable starting point and frequently used. A three-layer architecture is a special case with predefined layers for use cases that can be roughly divided into the three areas data, business logic, and presentation.

The requirement for ingestion (R_1) and storage (R_2) of heterogeneous data can be met with a storage concept such as that of a data lake which ingests data from a variety of data sources. The data sources mentioned in requirement R_1 are not static and may change in the future. For example, the number of data sources may increase. Contrary to that, a storage concept for heterogeneous data, as needed due to requirement R_2, is very unlikely to change. The probability of changes to requirement R_2 is low because a future change request for a fundamentally different data storage concept is unlikely unless other requirements, such as R_1, change as well. Due to the different probability for future changes to both requirements, their solution should be placed in different layers of the resulting architecture.

Requirement R_3 implies the storage of metadata. As such, a solution to this requirement may be placed on the same layer as the storage requirement in R_2.

Providing applications and results derived based on the data, metadata, and work activities (R_4) is an overarching requirement of R_1 and R_3. As it needs to interface with the storage of both data and metadata, it is affected by each change to these requirements. No other requirement is affected by this combination of requirements. This makes an implementation of R_4 on its own layer, decoupled from R_1 and R_3, a preferred solution to keep complexity down.

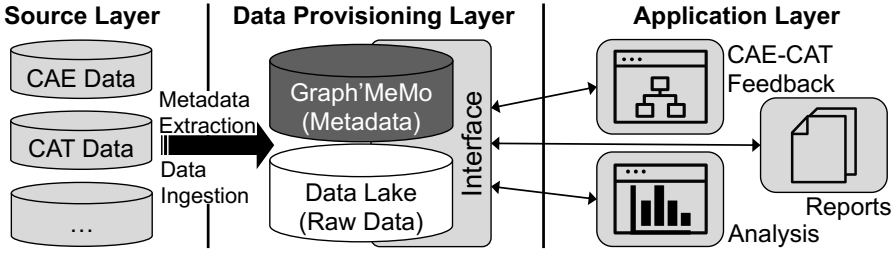

Figure 3.4: System architecture of a system to ingest, store, and provide data and metadata using Graph'MeMo [ZRKM21].

In a second iteration, both requirements and the architecture may be further detailed. A closer look at the requirement R_3 reveals that work activities and the relationships can be regarded as metadata. Thus, they are covered by requirement R_2. However, a concretization of requirement R_3 shows that a meaningful representation of work activities also necessitates associated metadata to describe these work activities. On the other hand, the relationships mentioned in R_3 necessitate links that connect existing data and metadata. The necessary flexibility and expressiveness for both may be high and exceed the capabilities of a heterogeneous data store, as required in R_2. Thus, this suggests that requirements R_2 and R_3 should be implemented on the same data-centric layer but in different components that separate data storage and metadata.

The resulting system architecture with three layers is illustrated in Figure 3.4. From left to right, these are the source layer, the data provisioning layer, and the application layer. The source layer contains all the data sources that the data provisioning layer may want to ingest data and metadata from. The number of data sources is flexible and may change over time, thus conceptually situated in its own layer. The data provisioning layer houses the heart of the system, which stores both heterogeneous data and metadata. Both data and metadata are stored in different components due to the different demands to their individual storage capabilities. The application layer is the home to all consumers of the data and metadata stored in the data provisioning layer. These may be reporting applications, analysis tools, or the use case presented in Section 3.1.2. Such applications are highly dynamic with their own set of very

different requirements. As such, they are decoupled from the otherwise stable data provisioning layer. To support these applications, the data provisioning layer contains an interface component for applications to use. This interface aims to provide a stable way for applications to access data, metadata, and work activities. Its powerful query mechanism supports future applications with yet unknown requirements.

3.4.2 Software Architecture for a Data Provisioning Layer

The previous Section 3.4.1.2 presented a system architecture with a data provisioning layer at its core (see Figure 3.4). This core layer embeds Graph'MeMo and provides the necessary functionality to implement the use case described in Section 3.1.2. Various software architecture patterns with different advantages and disadvantages are potential candidates to implement this data provisioning layer. These patterns have been evaluated in the course of a student project that has accompanied the work on this thesis [Boc20].

A naive client-server architecture implements the data provisioning part as server with an exposed interface for clients in the application layer. Communication takes place in a request-reply pattern and is a good match for synchronous usage scenarios between client and server. However, scalability and extensibility of the server is difficult, and ease of client support depends on the chosen communication method. A Service-Oriented Architecture (SOA) provides different features of the data provisioning layer as services that are then exposed to potential clients in the application layer. These services can be organized around the business logic and form isolated packages that ease scalability and extensibility. There exist industry standards to implement a Simple Object Access Protocol (SOAP)[6]. SOAP is a messaging protocol to send information for web services. The Web Services Description Language (WSDL) allows defining web services and specifies how to communicate with them, and Web Services Dynamic Discovery (WS-Discovery) defines a protocol to locate available services. Although this is not the only technology stack for SOA, they generally share a high complexity [Ric15].

[6]Simple Object Access Protocol: https://www.w3.org/TR/2000/NOTE-SOAP-20000508

A better fit for a software architecture for the data provisioning layer is a Representational State Transfer (REST) architectural style with hypermedia-driven microservices. It is built with a microservice approach that keeps each service limited to its dedicated business functionality [NMMA16]. Communication between client and server is layered and RESTful. Such a software architecture is scalable and extensible through the microservice approach [NMMA16]. The web-standard communication technology and REST style ease development of client applications due to the broad support for these technologies.

Figure 3.5 shows the resulting software architecture. Client applications send RESTful requests to an Application Programming Interface (API) gateway. The gateway routes these requests to the appropriate microservice suitable to handle the request. This way, the individual services do not need to be exposed to the public directly.

Services that handle metadata and data are located at the center. Notice that the data service, connected to the data store, is not directly accessed by client applications. Instead, client applications can store and request data through the REST API exposed by the API gateway. Graph'MeMo represents data sets as nodes and, thus, data is represented in the metadata structure. However, the storage for these data sets is separated from the metadata structure and allows using appropriate storage technologies for each individual data set. The REST API transparently handles retrieving data for client applications. Clients populate and query the metadata structure while the data service transparently stores and retrieves data sets associated with data container nodes. The ontology service handles Graph'MeMo's node types. It stores the ontology and handles requests concerning node types.

Aside of this core functionality, two more important aspects of such a software product need to be considered. One aspect concerns restricting access. When large amounts of data are pooled at one place, user management and authentication are necessary. Thus, a corresponding authentication service requires client applications to authenticate themselves before data and metadata operations can be used. The user service manages what operations an authenticated user can access.

The other aspect concerns compute-intensive tasks and compute resources. While the query interface allows client applications to fetch large amounts of

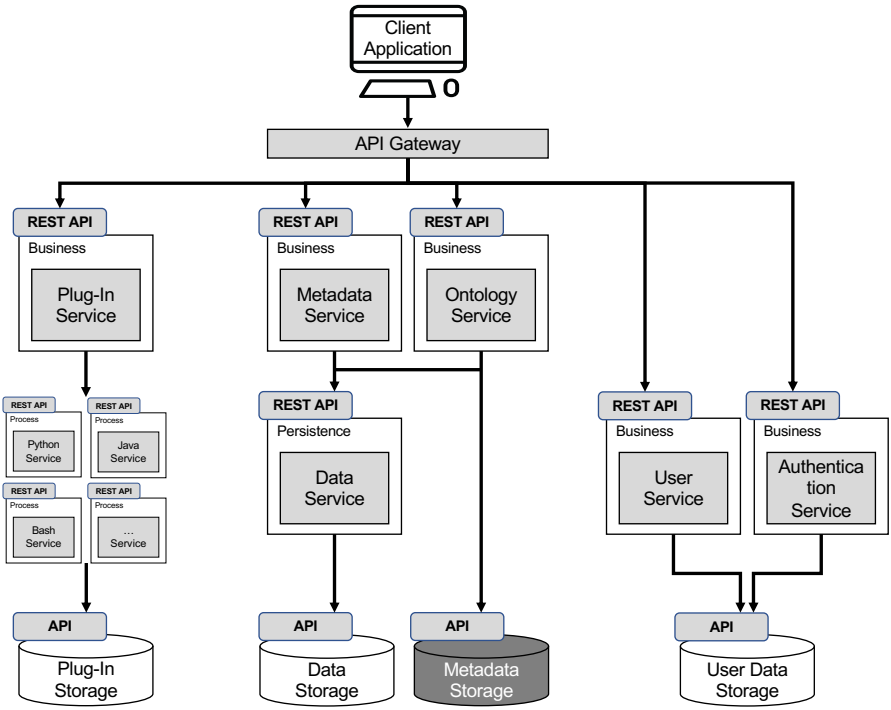

Figure 3.5: Software architecture for the data provisioning layer in Figure 3.4 using a REST architectural style with hypermedia-driven microservices.

data, these also need to be processed. In the case of a lightweight reporting application, the need for large compute resources in such lightweight clients may hinder the adoption of this system. At the same time, the implementation of this system is already equipped with large compute resources that it needs for itself. As such, it makes sense to make these compute resources available to lightweight client applications and allow them to offload compute-intensive tasks. Thus, a plug-in service exists that accepts program routines submitted from a client application. These routines enable direct access to the data and metadata and are executed within the compute resources of this system. The plug-in service then acts as an orchestrator for these submitted routines. Once

a routine is submitted, its execution and retrieval of results can be facilitated by the client application through subsequent calls to the REST API.

3.4.2.1 Implementation

The software architecture of the data provisioning layer is implemented based on a data lake for heterogeneous data storage and a graph database for the metadata structure.

The implementation uses a data lake for data storage due to the nature of heterogeneous and large data in virtual product development. A data lake is a concept for primarily, but not exclusively, storing raw data [Fan15]. It is generally associated with a large and scalable storage based on low-cost commodity hardware. For virtual product development data, the implementation requires a storage solution that additionally supports data having varying data structures that are not known in advance, e.g., in case of proprietary file formats. Multiple storage solutions with such capabilities are available. For example, Amazon S3[7] is a cloud-based storage solution that supports structured and unstructured data, is scalable, and fault-tolerant.

For this implementation, an on-premise approach is preferred and Hadoop Distributed File System (HDFS)[8] offers similar capabilities in a local deployment. To some companies, an on-premise solution may be desirable due to information exchange policies of their customers. Some data may also be restricted by local laws, e.g., export-controlled data such as military data that mandates storage within certain geographical regions. HDFS's distributed architecture allows scaling up storage capacity as needed, and it provides fault tolerance for both data and data processing. Fault tolerance for data is accomplished through configurable replication, while data processing functionality can be duplicated to achieve reliability. Another major benefit is Hadoop's extensive environment that allows adding more functionality if desired, such as data processing and analytics with MapReduce and Apache Spark[9]. Furthermore, Apache HBase enables real-time access to large, structured data, while Apache Hive provides

[7]Amazon S3: https://aws.amazon.com/products/storage/data-lake-storage
[8]Apache Hadoop: https://hadoop.apache.org
[9]Apache Spark: https://spark.apache.org

data warehouse capabilities and support for SQL queries. If cloud-based data storage is desired, HDFS can be deployed on all major cloud platforms.

The logical model of Graph'MeMo's conceptual model is a property graph Therefore, a graph database that supports the property graph data model is a straightforward solution to store this metadata structure. Multiple options for graph databases are available, many even support a data model that extends a regular property graph. JanusGraph[10] is a distributed and scalable implementation of a pure graph database. OrientDB is a database solution that supports multiple data models, including graphs. Azure Cosmos DB[11] and Amazon Neptune[12] are both cloud-based databases that support multiple data models, including property graphs. This implementation here uses an on-premise version of Neo4j as graph database that is specifically tailored to support the property graph model [Mil13]. It is a widely supported database, features scalability, high availability setups, and Atomicity, Consistency, Isolation, Durability (ACID)-compliant transactions.

The implementation utilizes the Relational Database Management (RDBMS) PostgreSQL[13] for user management and authentication. Plug-ins are stored on the local file system.

An advantage of a microservice architecture such as used here (see Figure 3.5) is that each microservice is independent in both programming language and platform. However, for this implementation, all microservices are implemented in Java using the Spring Framework[14]. This popular open-source framework eases the development of web applications and RESTful web services. The REST API exposed by these services provides CRUD functionality to client applications. For more complex queries, the API allows passing Cypher[15] queries directly to Neo4j. Cypher is the declarative query language of Neo4j that resembles Structured Query Language (SQL).

In general, the data provisioning layer ingests data from the source layer, stores them as files in the data lake, and generates appropriate metadata nodes that are stored in the graph database. A client application can also upload

[10] JanusGraph: https://janusgraph.org
[11] Azure Cosmos DB: https://azure.microsoft.com/services/cosmos-db
[12] Amazon Neptune https://aws.amazon.com/neptune
[13] PostgreSQL: https://postgresql.org
[14] Spring Framework: https://spring.io/projects/spring-framework
[15] Cypher: https://neo4j.com/docs/cypher-manual

```
MATCH (n:SpecificSimulationApp)-[:output]->(m:ResultData)
WHERE n.version > 2.5
RETURN m
```

Listing 3.1: Basic Cypher query with pattern matching and a predicate [ZRKM21].

data and assign a label as its type via the REST API, which then automatically stores the file in the data lake and creates a corresponding data container node in the metadata structure. Activity nodes can be created and added through another call to a corresponding API endpoint. Moreover, the API offers an endpoint to create contextual metadata nodes with their properties and label and to associate these metadata nodes to either data container or activity nodes. Further properties can be added, altered, or removed by updating metadata nodes. An API endpoint for relationships allows creating edges between two given nodes, delete edges, and list all incoming or outgoing edges of a node. To retrieve data, a client can traverse the graph-based metadata structure until it encounters a desired node. The client may then retrieve the properties and label of the node. In case of data container nodes, an API endpoint downloads the data files using the URI associated with these nodes. This happens transparently to the client. The web server automatically fetches and serves data files for data containers that are the result of requests.

Through the REST API, a client can also instruct the server to extract a subset of data and process it with a program routine supplied by the client application. For this purpose, the client supplies a plug-in to the server that contains the corresponding program code. Through Cypher queries, large sets of nodes can be retrieved with a single API call. Cypher queries may also be used for more complex queries than what the CRUD REST API offers directly, e.g., queries with filter predicates or pattern matching. Attributes of contextual metadata nodes can be used in search queries as well. As an example, a query may select all data container nodes that are the output of an activity node, representing a specific simulation application, but only for recent versions (`> 2.5`) of this application. This Cypher query is listed in Listing 3.1.

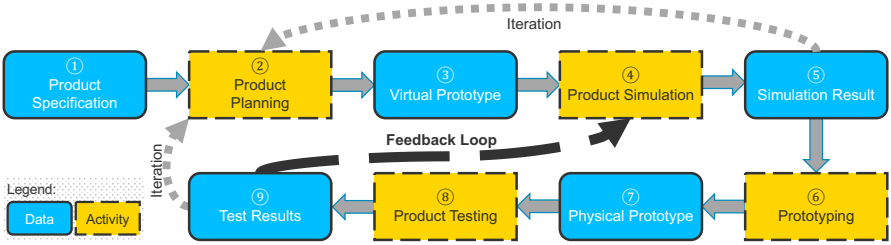

Figure 3.6: Simplified product development project with feedback loop between simulation data and testing data [ZRKM21].

3.4.3 Case Study: CAE-CAT Feedback

This case study demonstrates how the use case from Section 3.1.2 can be implemented based on an implementation of Graph'MeMo as previously described in Section 3.4.2.1. The use case intends to connect simulation data, and corresponding data from product tests, i.e., the data silos of CAE and CAT data, respectively. Such a connection may help to optimize simulation models through improvements based on analyses on historical testing data. Once implemented, this results in a feedback loop between testing data and simulation data, as illustrated in Figure 3.6.

The solution to this use case is implemented in the application layer of the system architecture described in Figure 3.4 with the Python programming language. Therefore, this case study implemented two client applications that perform the necessary operations to create a feedback loop between simulation data and testing data. The first client application is a data extractor and ingests source data into the data lake and describes them with appropriate metadata. The second client application is a data selector and queries fine-grained subsets of data that can then be used to improve simulation models.

The data extractor is responsible for ingesting the source data into the data provisioning layer and for creating the corresponding metadata nodes. It creates appropriate nodes and edges in the graph structure to describe the newly ingested data. Due to missing standards for the various data formats and metadata in virtual product development, no practical and mature approach exists in literature to extract relevant metadata from CAE files in a generic way. Therefore, the implementation of this data extractor is a domain-specific and

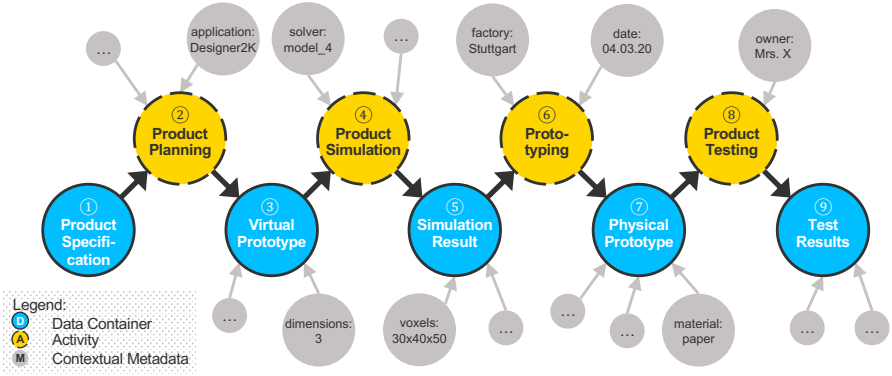

Figure 3.7: Instantiation of a graph structure representing data, activities, and metadata for a product development project as depicted in Figure 3.1 [ZRKM21].

tailored solution to this use case of a CAE-CAT feedback loop for the specific data at hand.

The data extractor is a modular framework comprised of several modules and adapters to extract metadata from the individual file types that occur in the given simulation data and testing data. Each module has a list of adapters that can perform the actual extraction of the metadata. Files are fed to modules that probe each file's content for structures they recognize. If the module recognizes a data structure for which it has an adapter, it calls this adapter to extract metadata from the file. Based on the extracted metadata, the module creates nodes, properties, and relationships between nodes through the REST API. An example of the resulting graph is illustrated in Figure 3.7. Our framework can easily support more file types by simply adding more modules or adapters. Existing adapters may be reused, if internal data structures are similar.

The data selector is responsible for querying fine-grained subsets of the previously ingested data, which is then used to optimize and validate simulation models. This optimization is achieved, e.g., by calibrating predefined constants in a simulation model. To obtain such constants, the data selector trains a Gaussian Process regression model on historical testing data. The data used for this training strongly influences the calibration constants it produces and, thus, the overall optimization of simulation models and the resulting quality

```
Select all (test results) for all (prototypes) (prototyped)
      with a (virtual prototype) from (product planning)
made of (cellulose)
and that were all (produced in) (factory Stuttgart)
and whose (virtual prototype) was (planned with) (Designer2K)
but only (version 11)
```

Listing 3.2: Pseudocode graph query with pattern matching and a predicate.

of their predictions. It is desirable to train different regression models for tailored calibration constants for very specific situations. That is, to optimize a simulation model used for products made of cellulose fibers, calibration constants are best obtained from a regression model that was trained with testing data of products also made of cellulose fibers. In practice, there are many more parameters than the material of the filter media. Therefore, the data selector queries testing data in dependence of many parameters and also the problem area itself: materials, product dimensions, intended purpose, and many more. All permutations require their own specifically trained regression model for optimized calibration constants.

The data selector client application uses the activities and contextual metadata of a simulation model to dynamically fetch appropriate testing data through the REST API. It downloads the selected test results from the data provisioning layer and trains a tailored regression model. Subsequently, this regression model can then determine calibration constants for specific simulation models. This way, the accuracy of the simulation is increased and aligned to the results from physical product testing.

Listing 3.2 shows a pseudocode query that selects data from a graph-style database. It illustrates how the data selector client application uses Graph'MeMo's metadata graph to select specific testing data sets from a graph instance such as the previous example in Figure 3.7. This pseudocode version is intuitively understandable, even for domain experts without background in computer science. The same query in the Cypher query language is shown in Listing 3.3 and bears visible resemblance to the pseudocode version, even for domain experts.

Both client applications run periodically. The data extractor is triggered by the testing data source. It then loads data into the system and subsequently

```
MATCH (t:TestResults)<-[*]-(pp:PhysicalPrototype)<--(pr:Prototyping)
      <-[*]-(vp:VirtualPrototype)<--(pp:ProductPlanning)
WHERE (pp)<--({material: "cellulose"})
AND   (pr)<--({factory: "Stuttgart"})
AND   (vp)<--(:Prototyping)<--({application: "Designer2K"})
AND   (vp)<--(:Prototyping)<--({version: 11})
```

Listing 3.3: Basic Cypher query with pattern matching and a predicate.

triggers the data selector to retrain those tailored regression models whose training data set changed. This ensures that with more testing data, the regression models are automatically retrained and new, optimized calibration constants are ready when needed. Computer simulations performed with these improved simulation models result in data with higher accuracy. Thus, resulting in an automated feedback loop between testing data and simulation data.

3.4.4 Discussion

Section 3.1.3 compiled challenges for data and metadata management within virtual product development. Based on the case study in the previous Section 3.4.3, this section discusses how Graph'MeMo embedded into the system architecture in Section 3.4.1.2 addresses these challenges. First, the general properties are addressed, then the individual challenges are discussed.

The structure and availability of data and metadata, as well as the work activities used to create these data in product development projects, vary greatly over time. One reason is that vendors of proprietary applications regularly change the amount and structure of produced data through major version updates. Skipping application updates is not feasible. The way domain experts process their data changes over time as well. Metadata provided by human users is often not consistent, leading to inconsistencies that require ad-hoc metadata updates at later points in time.

The data lake component of the data provisioning layer provides **flexibility** and enables to store data from CAE (simulation data) as well as results from CAT (testing data) together in their respective raw file formats. Data can be attributed with the metadata that is currently available without the need to conform to a predefined schema, further demonstrating flexibility. In the

future, existing data can be updated ad-hoc with additional metadata without schema adjustments. The abstract view of Graph'MeMo on the data, mapped to nodes in a graph, allows **generically** representing all data of virtual product development. Graphs are a **simple** structure that domain experts can intuitively understand with little training [AG08]. This was confirmed by the application of Graph'MeMo at the industry partner.

MC_1 **Isolated data silos with heterogeneous data sets**: The first challenge highlights that an approach to managing data and metadata of virtual product development needs to be able to represent all kinds of heterogeneous data found in product development projects. For this reason, Graph'MeMo provides a level of abstraction that allows representing data of different domains in the same way.

The schemes or low-level structures of proprietary file formats are rarely known to users in virtual product development. Domain experts usually work with their data through graphical applications and rarely need nor understand the underlying low-level data structures. Graph'MeMo takes this into account and represents data in a data container as single node in a graph. This limits the ability to directly represent more specific details such as the low-level structure or schema of data files. As such, the chosen abstraction level of the metadata model is tailored to the work habits of domain experts. Nevertheless, note that users are still able to fetch information about low-level file formats via the URI that is associated with a data container node and that refers to the underlying storage location. Furthermore, the structure of the data represented with a data container may be described through associated metadata if necessary.

The case study successfully connected data from multiple data silos. The implementation of the case study loaded the data into the data provisioning layer by creating the appropriate nodes in the Graph'MeMo graph. All different formats and structures of the data were handled the same way and no information about low-level data structures was necessary. The implementation of the case study connected related data sets and attributed them with metadata independent of their format and structure. Thus, the same implementation was able to handle the heterogeneous data of virtual product development.

Figure 3.7 exemplarily shows how data from units such as product planning (step 3), product simulation (5), and product testing (9) are integrated into

one model. So, Graph'MeMo is capable of representing all data across boundaries of different organizational units. Likewise, domain experts do not need knowledge about the data structures to use Graph'MeMo for retrieving data. The uniform access to all data allows domain experts to use Graph'MeMo to access data sets that were previously separated in data silos, e.g., by following connections between data. They can explore newly integrated data and extract knowledge that may be gained by interrelations among different data sets.

MC_2 **Activities have to be represented as first-class entities with their own metadata**: The second challenge concerns the work activities performed by domain experts and their representation as part of the metadata structure as first-class entities. For domain experts, the work activities they perform in development projects are more important than the data sets these activities produce or consume. Domain experts are more likely to remember the activities they or their colleagues carried out than the data sets the activity resulted in. Therefore, work activities as first-class entities improve the discoverability of data for domain experts. Together with their own respective metadata, they improve comparability and reproducibility of data sets, since the direct influence of these activities can be explicitly traced and analyzed. Thus, there is a need to represent them as first-class entities instead of representing them as mere annotations to data sets.

In this case study, work activities were represented as part of the graph structure created by the implementation. Comparable to data sets, each work activity was abstracted by a node in the graph structure and associated with metadata. In particular, the implementation used the same means to attribute data and work activities with contextual metadata to further describe these two kinds of entities. Semantic metadata can freely be attributed to data and work activities because the same mechanism is used in both cases. The level of abstraction also allows viewing all kinds of data, including work activities, in a unified view through an interconnected structure, further supporting MC_1. With activities and their metadata as first-class entities, the implementation of the case study has the necessary metadata for fine-grained data selection. That is, it can select test results from products designed with a specific material and even with a specific version of a CAD application.

The result is illustrated in the example instantiation of such a graph structure, Figure 3.7. It shows data, relevant work activities, and contextual metadata in one connected view. So, Graph'MeMo is capable of representing work activities as part of the metadata structure as first-class entities. Domain experts use them to find the data sets they need, and assess comparability and reproducibility of the data sets they are associated with.

MC_3 **Relationships between data and activities have to be represented**: The third challenge expresses the need to explicitly describe relationships between data and work activities. Domain experts are usually familiar with such a structure because the dataflow it represents corresponds to the way domain experts work in their product development projects. So, they can intuitively navigate such a metadata model and easily find desired data. With work activities as first-class entities and relationships between data, the metadata structure represents the dataflow between these previously isolated data sets within product development projects. Moreover, the accurate representation of both the dataflow and the context of data in product development projects provides information about data lineage. Domain experts, i.e., may use this information to examine the way data was manipulated or produced in order to assess reproducibility of results and the quality of data. In addition, the dataflow in virtual product development itself can be the subject of analysis to gain additional insights into projects. For example, the progress of projects can be analyzed to find patterns that influence key metrics, such as the duration and costs of development projects.

In this case study, associations between data containers and work activities represent the flow of data between those elements. Graph'MeMo is implemented with a property graph model, where these elements are expressed as nodes in a graph. Nodes representing data containers may be connected to nodes representing work activities with directed edges that indicate the direction of the dataflow. A work activity may produce multiple data sets and, thus, have multiple outgoing edges. Likewise, a work activity may work with multiple data sets and, thus, have multiple incoming edges. This creates a causal order among all connected data sets and activities, which is then utilized to discover and select data. The implementation of this case study uses this

Table 3.3: Comparison of related work to Graph'MeMo. ● = fulfilled; ◑ = partially fulfilled; ○ = not fulfilled [ZRKM21].

	MC_1	MC_2	MC_3
Generic Metadata Models [HSG+17; QHV16; SSF+19]	●	○	○
Specialized Metadata Models [MSF+10; VFP+05; VEC04]	◑	○	○
Workflow Modeling Techniques For example, BPMN [Obj14]	◑	◑	●
Commercial Computer Applications	◑	○	○
Graph'MeMo [ZRKM21]	●	●	●

order to select such data sets that were produced by specific work activities and such data sets that were produced based on other specific data sets.

The associations in Graph'MeMo (see Section 3.3.1) are capable of representing relationships between data containers and work activities. The graph structure shown in Figure 3.7 and its connections between data containers and work activities closely resemble the corresponding workflow of the product development project illustrated in Figure 3.1. The metadata structure even extends this notion of a project-specific dataflow with additional contextual metadata, which further describes data containers and work activities. So, the explicit relationships between data and activities as well as the contextual metadata together provide meaningful descriptive information. All this provides domain experts with powerful and familiar means to explore and select data, even across organizational units.

Table 3.3 summarizes Graph'MeMo compared to the previously discussed related work in Section 3.2 in light of these challenges.

3.5 Summary

This chapter proposes a metadata model that offers a tailored approach to data management in virtual product development by addressing three domain-

specific challenges. It eliminates data silos by providing a connected view on all heterogeneous data, metadata, and activities in virtual product development. Work activities are explicitly modeled as first-class entities alongside data. This allows describing them with metadata and improve comparability and reproducibility for domain experts. Furthermore, the metadata model explicitly represents relationships between data and activities that reflect the dataflow of product development projects. This provides valuable context that makes the metadata model familiar to domain experts. The presented prototypical implementation for a real-world use case shows that this empowers domain experts to interpret, select, and analyze their data. For instance, it facilitates them to retrieve CAT data that is necessary to increase the accuracy of CAE simulation models.

Chapter 4

Integration of Domain Experts into Data-driven Surrogate Modeling

In virtual product development, simulating products is essential. However, simulations can be time-consuming and costly. Surrogate models can help address these challenges, as they offer a simplified representation of an experiment or simulation that would otherwise be difficult, time-consuming, or expensive to compute or observe. These surrogate models serve as approximations and are, thus, associated with a certain degree of error [JZS20].

Surrogate modeling can be performed manually, by creating simplified or less complex simulation models that are faster and more cost-effective due to a simplification of physical considerations [JZS20]. Alternatively, a data-driven approach can be used, involving the automated creation of predictive models based on historical simulation data or real-world testing data. The latter approach enables automation and can save time compared to manually creating simplified models. Moreover, a data-driven surrogate model can be easily updated when new or additional data becomes available, continually

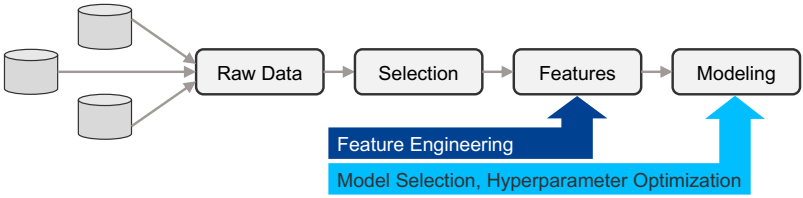

Figure 4.1: The points of interest for the proposed approach: feature engineering and modeling.

improving its quality. There are three major components to creating useful data-driven surrogates: data selection, feature engineering, and model training.

A flexible selection of data is essential for this process [ZRKM20]. Due to the time-consuming nature of these simulations, the amount of available data is limited. Selecting suitable training datasets from these little data has a high impact on prediction quality. This requires domain experts who understand the physics behind the simulations that are to be approximated. With a solution such as Graph-based Metadata Model (Graph'MeMo), domain experts can apply their domain knowledge and discover, as well as select, the most appropriate datasets for training a particular surrogate model [ZRKM21].

Feature engineering is an umbrella term for several tasks, such as feature extraction, feature construction, and feature selection. They highly depend on domain knowledge and greatly influence the prediction quality of the models [WSR+20; HZC21]. Although some automation exists for feature engineering, the resulting surrogate models often no longer rely on physical foundations [TWP+19; WSR+20]. Algorithms usually do not understand any physical laws and, e.g., calculate values regardless of their unit of measurement. To address this problem, domain experts can be incorporated into the process. In this chapter, domain experts are involved in the data-driven creation of surrogate models through participation during feature engineering.

Automated model training and corresponding hyperparameter optimizations are well established in literature [HZC21]. Accordingly, these technologies are applied during model training.

To this end, this chapter describes a generic approach for data-driven creation of surrogate models that allows updating them when new or additional data

becomes available, continually improving their quality. The approach integrates knowledge from domain experts, enhancing the reliability and decreasing the error of the resulting models. Additionally, the approach incorporates existing techniques from automated machine learning and interactive machine learning [Hol16]. Figure 4.1 shows the two main points of interest for the proposed approach. Feature engineering benefits from integrated domain knowledge and partial automation. Model selection and hyperparameter optimization benefit from a fully automated strategy. A case study demonstrates the application of this approach and evaluates its results. Consequently, the physical correctness of the surrogate models is ensured, while still benefiting from an automatable, data-driven machine learning pipeline. This empowers domain experts and promotes the democratization of data science and analytics.

This chapter presents contribution C_2 and addresses research challenge RC_2. The contents of this chapter are an extended and revised version of an earlier publication by the author [ZRK+24].

The remainder of this chapter is structured as follows: A use case and derived challenges for creating data-driven surrogate models with physical foundation are presented in Section 4.1. Section 4.2 discusses related work. The approach to data-driven surrogate modeling with integrated domain knowledge is presented in Section 4.3. Section 4.4 describes the implementation, the case study, and the results of the evaluation, finished with a discussion. Section 4.5 concludes and lists future work.

4.1 Use Case: Long-Running Computer Simulations

This section presents a use case for surrogate models for long-running computer simulations. Section 4.1.1 explains the overall objective of this use case. Section 4.1.2 describes the simulation system to be modeled by a surrogate and Section 4.1.3 describes the data available in this use case to create a data-driven surrogate. Based on this, Section 4.1.4 derives challenges that arise in a conventional approach.

4.1.1 Objective

In this use case, a simulation examines the so-called *medium* in a filter element. A medium is the material in a filter that separates unwanted substances, e.g., dirt particles during the filtration process. It may consist of various materials, such as cellulose, plastic or even activated carbon. Based on a virtual prototype of the medium, the basic properties of the medium are first assessed through a computer simulation. This simulation is a lifetime simulation that models the entire lifespan of the medium, revealing all characteristic performance features. Such a simulation is very time-consuming.

The virtual prototype subjected to the simulation is sometimes based on a real material that has been digitized using an X-Ray Computed Tomography (XCT) scan, but it may also be generated completely virtually as a 3D Computer-aided Design (CAD) model. Virtual generation of media is particularly useful for creating and simulating entirely novel media that do not yet exist in the market. If the material's characteristics are found to be better compared to existing media, the exclusive production of this medium can be initiated. In this way, the industrial partner of this thesis, MANN+HUMMEL GmbH, gains an innovative advantage at a fundamental stage of filter development. Optimizations or the creation of novel media can target different characteristics, depending on the application of the filter to be created with the medium.

The objective of this use case is to create a data-driven surrogate model to replace the long-running simulation at the early stages of development. The surrogate model should provide acceptable and reliable prediction of target values through regression analysis with significantly reduced runtime. If the surrogate model's approximation is sufficiently good, the industrial partner can quickly examine and iterate new virtual prototypes to find optimal new media. The long-running simulation is then used at a later, more concrete stage of product development. For example, the surrogate model can be used to create a reduced shortlist of prototypes for closer examination and only the most promising prototypes undergo the long-running simulation, possibly only to conclusively confirm the predictions and expectations. This is only possible if the approximation model generalizes adequately.

Figure 4.2: 3D model of the virtual prototype of a filtration medium and deposited particles of different sizes. Rendering provided by MANN+HUMMEL GmbH.

4.1.2 The System to be Modeled

The simulation system for which a surrogate model is sought is a problem of the engineering domain, specifically virtual product development. This problem is addressed by simulation engineers who simulate the entire lifespan of the medium with computer software. The input is a virtual prototype that goes down to the fiber level, making it highly detailed.

The simulation involves a standardized mix of particles flowing through the medium until it is no longer possible due to blockage. Figure 4.2 shows such a medium, which is currently being simulated and is already partially filled with particles. When the medium is physically blocked because it is completely filled with particles, the simulation ends. At this point, several interesting key metrics can be evaluated. One such metric is pressure drop, which is the pressure difference between the inflow and outflow sides at various points in time during particle loading. Another metric is dust holding capacity, which is the particle volume within the filter medium. Depending on the medium, blockage occurs at different points in time, meaning the simulations have varying durations.

4.1.3 Simulation Data

A dataset provided by the industrial partner, MANN+HUMMEL GmbH, contains previously computed simulations. The simulation data, from which surrogate models are to be trained, include 112 fully simulated virtual prototypes. This is not a large amount of data, but due to the time-consuming nature of the simulation process, it is not feasible to increase the amount of data significantly.

Each simulation of a virtual prototype is a time series of varying length. The simulation runs in discrete time steps until the target state (blockage) is reached. Consequently, for each virtual prototype, data for all time steps of the simulation are provided. This dataset of computed simulations contains between 6 and 13 time steps, depending on the prototype.

The dataset also resolves the virtual prototypes in their spatial dimensions. The spatial dimension is a discretized excerpt of the data, as the raw amount of data would otherwise be overwhelmingly large. So, instead of having a record of data for each point (x, y, z) in three-dimensional space, the values have been aggregated along two dimensions to so-called layers. The depth of the filter medium is sliced into discrete layers, with each layer being of equal fixed size and spanning a rectangular subspace. Data for a layer is the average of all values across the entire layer. The layers all have equal size for comparability. However, the prototypes are not all the same size, meaning some have more layers than others. Thus, the simulated prototypes have between 13 and 16 layers. Layers are numbered sequentially, ascending toward the end of the medium, with the end being on the outflow side.

In addition to an identifier for the virtual prototype, the time step, and layer, the following features are included in the data:

- Solid volume fraction: the percentage of the fiber volume in the total volume of a layer. This value remains constant throughout the simulation, as the fibers do not change.
- Pore size in 3 deciles: pores are the empty spaces in the volume that are neither fibers nor particles. Their size is characterized in three deciles. For example, the first decile of pore size represents the size in micrometers below which 10% of all pores are found. That is, 10% of all pores are smaller than or equal to the value of this decile.

Table 4.1: Features and approximate value ranges of the dataset.

Feature	Unit	Value Range	
		min	max
Solid volume fraction	$\%$	0.01	0.50
Pore size, 1st decile	μm	2	100
Pore size, 5th decile	μm	2	200
Pore size, 9th decile	μm	10	300
Collected particles	$\%$	0	1
Particle size, 1st decile	μm	0	20
Particle size, 5th decile	μm	0	50
Particle size, 9th decile	μm	0	60
Pressure drop	$\Delta p\, Pa$	0.1	200 000

- Collected particles: the percentage of the volume of deposited particles in the layer. This value is not constant and changes throughout the simulation as more particles are deposited.
- Particle size in 3 deciles: the size of particles deposited in the filter medium, i.e., accumulated particles. They are characterized in three deciles, analogous to pore size.
- Pressure drop: the pressure drop represents the difference between the static air pressure in front of the filter medium (inflow side) and the pressure after the respective layer for which the value is given. The unit of measurement is Pascal. This value describes how strongly the filter medium, which is increasingly blocked by particles, hinders the air flow or increases the pressure.

Table 4.1 lists the approximate value ranges for each feature along with their units of measurement. It shows that multiple features, such as the three pore sizes, could be used in feature construction without violating physical laws because they have the same unit and measure the same physical property. However, the unit of measurement alone is unsuitable to make that assessment. A domain expert is needed to make that decision because it not only depends on laws of physics, but also on the specific use case and, potentially, on many other considerations a data scientist may not be aware of.

4.1.4 Challenges

The data and the goal of generating data-driven surrogate models present challenges that need to be addressed. Two challenges are commonly encountered in this domain and constellation:

C_D Data Challenge: First, the simulation of virtual prototypes is time-consuming, resulting in a limited number of simulated prototypes and, thus, small datasets. In addition, the datasets often have a small number of features. Moreover, due to the underlying material science and physics, datasets of filter media of different materials cannot be mixed. Accordingly, a surrogate model for a filter medium of a specific material requires a dataset of the same material. This decreases the number of suitable training data even further. In such cases with only small amounts of data, a highly specific dataset selection is important and may help ensure the accuracy and trustworthiness of the predictions [ZRKM20; HRT+23].

Second, the required granularity goes even further in reality. Surrogates may need to be separated not only by materials, as in the given example, but also based on the manufacturing process and processing plant of a material. This is because different machines and processing parameters may lead to different and significant changes in the same material. This adds further complexity for the selection of the optimal training datasets.

C_K Knowledge Challenge: The absence of knowledge-based feature engineering by domain experts significantly complicates the surrogate modeling process. The data contains complex physical intricacies that may not be fully understood by data scientists or automated tools [BWK+22; TWP+19]. These complexities involve interactions between various factors, such as material properties, manufacturing processes, and operating conditions, which can significantly impact the performance of the filter media. This challenge necessitates close collaboration between domain experts and data scientists to ensure that critical domain knowledge is effectively incorporated into the modeling process [LPS18]. Overcoming the knowledge challenge is crucial for the successful development and deployment of data-driven surrogate models in this specific engineering domain.

4.2 Related Work

Domingos [Dom12] highlights feature engineering as the most interesting aspect of machine learning. It requires a mix of technical knowledge, intuition, creativity, and a certain level of domain expertise. Guyon and Elisseeff [GE03] claim that a subset of features often yields better results than the entire original feature set. Feature engineering can improve the speed and decrease the error of regression analysis. Depending on the task and objective, there are different types of feature engineering, such as feature selection and feature construction.

Feature selection can be accomplished using filters, wrappers, and embedded methods [PCR19]. Filter-based selection is based on correlation metrics between features. Wrappers evaluate the quality of feature subsets after training, making this approach slower but providing more accurate results. Embedded methods optimize a function that assigns weights to features, aiming to minimize the impact of a subset of features. This allows for the partial inclusion of features.

Feature construction involves creating new features based on existing ones through various operators, such as multiplication, addition, or applying other functions such that $new\_feature = g(f1, f2, ...)$. Guyon and Elisseeff [GE03] suggest that an apparently less useful feature, when added, can sometimes drastically improve prediction. A potential strategy is to generate as many new features as possible and then reduce them using selection. Tackling this NP-hard problem, Krawiec [Kra02] proposes an approach of selecting logical and arithmetic operators, creating a finite number of new features through randomized operators with the original features. However, such an approach does not consider laws of physics.

Tsai, Cai, and Wu [TCW98] suggests to use residual plots to identify further need for feature engineering. Certain patterns may indicate specific issues, like the outward-opening funnel or the double arc, both of which show a non-constant error variance. The authors recommend addressing these issues by applying appropriate transformations to the features or the target value. Such transformations should ideally incorporate a significant amount of domain knowledge to achieve the best results.

Other approaches integrate domain knowledge into the training process of surrogate models. For instance, Hao et al. [Hao+21] propose a language to describe conditions between variables of the training data. These conditions may be used to improve the training of surrogate models. However, the proposed language is limited in its expressive power, so that only simple conditions may be specified. Yang et al. [Yan+21] propose an approach to integrate Partial Differential Equations (PDEs) into the structure of a neural network to approximate the simulation models that are described by these PDEs. This approach, however, focuses on a specific problem and is restricted to a specific kind of neural network. In fact, none of these approaches integrates domain knowledge into feature engineering.

4.3 Data and Domain Knowledge Driven Approach

This section presents an approach to data-driven surrogate modeling that integrates domain knowledge into the machine learning pipeline, specifically feature engineering. Before such an approach can be used, suitable training data are required. The training data are crucial for the quality of a machine learning model. In the domain of virtual product development this is already a complicated task, which requires domain knowledge [ZRKM20; ZRKM21]. Therefore, a way is needed that allows domain experts to make very specific and fine-grained selections of data. The graph-based metadata model Graph'MeMo introduced in Chapter 3 provides these properties. Listing 3.2 and Listing 3.3 show examples of how such a selection can be made, which, for instance, only selects filter media of a specific material and manufacturing location. The following approach assumes that an appropriate selection of training data has already taken place and can now be used by the approach to train surrogate models.

Section 4.3.1 provides an overview of a data-driven approach with integrated knowledge from domain experts and describes the steps involved. This establishes a machine learning pipeline capable of repeated, automated execution with integrated domain knowledge, e.g., when new data is available. By including domain experts into the data-driven process, the approach effectively combines automated machine learning and interactive machine learning with

domain expertise, merging the advantages of both strategies. Note that the following Section 4.4 will then take a closer look at the approach and further detail it with implementation details and real-life experience in a case study based on the use case presented in Section 4.1.

4.3.1 Approach

A purely data-driven approach has drawbacks, particularly concerning correctness and compliance with the underlying physical science that the surrogate aims to approximate. Therefore, domain experts play an integral role, offering their valuable knowledge and experience to improve the quality of surrogate models. To effectively incorporate domain knowledge into the machine learning pipeline, this section first defines specific types of expertise. Based on a classification from von Rueden et al. [vRMB+23], two distinct categories of knowledge fit for this approach:

1. Scientific Knowledge: Domain experts contribute their understanding of the valid numerical operations for feature engineering. This knowledge is derived from the principles and laws of physics and engineering that the domain experts have studied and applied in their fields. The most appropriate way to represent this type of knowledge is through the use of algebraic equations and mathematical functions, which can effectively convey the scientific concepts and relationships that underpin the expert's understanding.
2. Expert Knowledge: Domain experts offer valuable insights into which features should be considered for analysis, which are crucial for accurate modeling, and which can be disregarded during the feature engineering process. This type of knowledge is rooted in the expert's intuition and experience, making it less formal and more flexible than scientific knowledge. To capture the nuances of expert knowledge, human feedback is the ideal representation method. This approach allows for the necessary informality and a degree of uncertainty that is vital for conveying the expert's intuitive understanding of the subject.

Figure 4.3 illustrates the data-driven approach to surrogate modeling with domain experts integrated into the machine learning pipeline. The approach

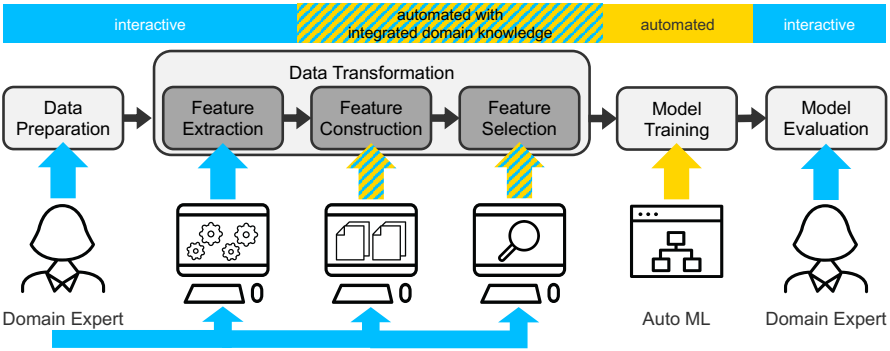

Figure 4.3: Steps of a machine learning pipeline with interactive and automated steps.

includes four distinct steps: data preparation, data transformation, model training, and model evaluation. The yellow arrow indicates where existing automations apply. Blue arrows depict knowledge of domain experts integrated into steps of the machine learning pipeline. Steps shaded in both colors are comprised of both, domain experts and their knowledge integrated into automations.

These four steps are based on the well-known Knowledge Discovery in Databases (KDD) and Cross-industry Standard Process for Data Mining (CRISP-DM) processes and can be briefly summarized as follows: Data preparation involves basic data engineering tasks, such as data cleaning, handling missing values, and data normalization. During data transformation, various techniques are applied, such as feature engineering, feature scaling, and dimensionality reduction, to ensure the data is in a suitable format for model training. Model training includes selecting the most appropriate model configuration, training the machine learning model, and optimizing hyperparameters to improve performance. In model evaluation, the trained model's performance is assessed against business requirements, including its error, generalizability, and computational efficiency. The individual steps with regard to this approach are described in more detail below:

Data Preparation: Data preparation involves cleaning and organizing raw data. This step addresses missing values, removes duplicates, handles out-

liers, and converts data types. For example, data preparation may involve removing incomplete entries, standardizing date formats, and encoding categorical variables such as product categories into a machine-readable form. In this step, domain experts contribute by identifying specific data cleaning and pre-processing requirements, based on the unique characteristics of the input data. They can identify the relevant data for a given task and, e.g., correctly categorize if spikes in the data are outliers or valid values. Their deep understanding of the underlying processes, data sources, and potential biases enables them to make informed decisions that enhance data quality.

Feature Extraction: If the raw data is not in the form of numerical features that can directly be used to train a model, such features must first be extracted from the raw data. Thus, depending on the data, feature extraction may be optional. Domain experts can recommend appropriate feature extraction techniques tailored to the specific type of raw data, e.g., images or spatial data. They can suggest suitable methods for extracting meaningful information from these data that are relevant to the use case. For example, the raw data from computer simulations of filter media may include velocity fields, pressure distributions, and particle paths. From these data, engineers can extract features such as the pressure drop across the filter media, or the filtration efficiency over time. The expertise of domain experts helps to identify relevant features and prevent the extraction of irrelevant or redundant features, streamlining the subsequent modeling process.

Feature Construction: Feature construction in machine learning involves creating new features from the existing features to improve the performance of a trained model. By transforming and combining existing features into more informative indicators, feature construction helps algorithms to understand the data better and make more accurate predictions. Feature construction can be performed in a data-driven fashion to automatically generate new features from existing data. However, data-driven construction of new features may generate violations of physical laws because automated approaches may not be aware of specific rules or nuances in domain-specific data. For example, an automated feature construction algorithm might not recognize different units of measurement among features. Though computationally similar, the algorithm may not know that one feature represents pressure in Pascal, while

another indicates the diameter of particles in micrometers. To address this limitation, it is necessary to integrate domain experts and their knowledge into the approach [LPS18]. Domain experts' familiarity with the domain-specific relationships between features allows them to guide the development of new features based on established scientific principles and laws of physics.

Therefore, this step constructs new features with a combination of automated methods and input from domain experts. These experts may provide algebraic operations or mathematical functions that may be used to construct new features. They create a compatibility matrix, which contains each feature as a row and as a column. Figure 4.4 shows an example of such a matrix for three features n_0, n_1, and n_2. The matrix's position (n_2, n_1) contains a binary algebraic operator or mathematical function f which is to be applied as $f(n_2, n_1)$. The matrix is not commutative, so a function at (n_2, n_1) does not apply to (n_1, n_2). If there is no function for a pair of features, this pair is not suitable for constructing new features. Finally, all combinations of features for which an operation is defined in the matrix can be constructed automatically. The fact that domain experts have specified the functions in the matrix ensures that the constructed features are not only mathematically sound, but also have real-world relevance. This ultimately leads to more accurate and interpretable surrogate models.

$$\begin{array}{c} \\ n_0 \\ n_1 \\ n_2 \end{array} \begin{array}{c} \begin{array}{ccc} n_0 & n_1 & n_2 \end{array} \\ \begin{bmatrix} & & +,- \\ +,g & & \\ & f & \end{bmatrix} \end{array}$$

Figure 4.4: Compatibility matrix of possible functions for automated feature construction based on three existing features.

Feature Selection: Feature selection is the process of identifying and selecting a subset of relevant features and thereby eliminating irrelevant or redundant features for use in model training. It is important because it can improve model performance by reducing overfitting, enhancing generalization, and decreasing training time. Feature selection can be automated through various methods.

For example, there exist simple filtering methods, feature importance ranking, and sophisticated techniques such wrapper methods [HKV19]. The wrapper method evaluates subsets of features by actually training a model on them and measuring the model's performance. It iteratively selects or excludes features based on their contribution to the model's predictive power. Different methods have different advantages, and it is prudent to use multiple methods as a composition, e.g., an initial filtering with a subsequent wrapper.

In this step, a combination of automated methods and input from domain experts is used to select subsets of features for model training. Domain experts can contribute to the selection process in several ways: a) identifying features that must be included during training, b) specifying features to be excluded, and c) assigning weights or ranks to features, allowing subsequent algorithms to prioritize features based on this information. Ideally, all three ways are combined. That is, domain experts create lists with must-have features, features to be excluded, and a weighting according to their expertise. Both filter and wrapper methods for feature selection can use this input to include, exclude, or weigh features with human expertise, rather than using purely statistical metrics. This collaborative method ensures that the selected features are both mathematically sound and domain-relevant, leading to more accurate and interpretable models.

Model Training: At this point, the presented approach established a machine learning pipeline that is enriched with domain knowledge. Now, the fully automated parts take over and, with the input from domain experts fixed, the pipeline can be re-used on demand. For example, when more data is available in the future, the existing pipeline with the pre-recorded input from domain experts can be executed again.

This step employs a fully automated machine learning approach, focusing on model selection and hyperparameter optimization. This involves training various models and comparing the results in order to find the best model for the intended purpose and the available data. In addition, the numerous hyperparameters of the models are iteratively optimized to maximize the performance of the desired results. Automated machine learning selects appropriate models and optimizes hyperparameters based on various metrics, such as error, ac-

curacy, or precision. This automation of the model training streamlines the training process, leading to more efficient and effective surrogate modeling.

Automated machine learning approaches often achieve results as good as or better than humans in model selection and hyperparameter optimization [HKV19; HZC21]. Therefore, it is reasonable to perform the model training step using automated machine learning tools or frameworks, as they can quickly explore a vast range of models and hyperparameters. Such tools or frameworks can be fully automated and data-driven, and fine-grained datasets can be queried from systems, such as Graph'MeMo.

Model Evaluation: Once the automated model training is complete, a domain expert then evaluates the results to validate their satisfaction of business demands, ensuring the model's suitability for the intended purpose. The inclusion of domain experts in the evaluation process ensures a comprehensive assessment of the model's performance. They can help interpret the results in the context of the specific domain, the business case, identify potential areas of improvement, and provide recommendations for further refinement of the model. Domain experts can evaluate a trained model by examining its predictions for consistency with known principles and empirical knowledge in their field. They can also assess the model's performance on specific cases that are critical or challenging in their domain, ensuring it aligns with realistic expectations.

The resulting approach combines the advantages of a data-driven, automated process with domain expert knowledge, making it compatible with industry requirements and enhancing the model's overall performance and reliability. The following Section 4.4 describes a prototypical implementation and demonstrates this approach with a case study.

4.4 Implementation and Case Study

This case study demonstrates the application of the proposed approach to data-driven surrogate modeling with integrated knowledge from domain experts, addressing the use case from Section 4.1 and challenges from Section 4.1.4. The objective is to perform regression analysis on a given virtual prototype to predict the outcome at the end of the computer simulation based solely on

the initial state. As a reminder, the dataset presents challenges for machine learning algorithms such as a low number of data points, few features, and potentially complex relationships between values.

Section 4.4.1 details the implementation of the data-driven surrogate modeling approach with integrated knowledge from domain experts. Section 4.4.2 presents the case study and an explanation of how the implementation addresses the use case. Section 4.4.3 describes the results achieved with the proposed approach and ends with a discussion of the results and a reflection on the challenges.

4.4.1 Implementation

The implementation of the proposed approach is written in Python 3.7 and leverages existing machine learning frameworks, such as scikit-learn [PVG+11] and other well-established libraries, such as NumPy [HMvdW+20] and pandas [McK10]. It implements a machine learning pipeline, which is structured into the consecutive steps of the approach presented in Section 4.3.1 and illustrated in Figure 4.3. The implementation includes a custom-tailored automated machine learning component that implements the proposed approach and logs necessary data for the evaluation. This particularly refers to the components for the feature construction and feature selection steps because these combine the knowledge of domain experts with automation. Parts of the implementation have been created in the course of a student project that has accompanied the work on this thesis [Nie20].

Automated feature construction is implemented by combining or otherwise aggregating existing features. The implementation forms all possible n^2 pairs of two of the existing n features and calculates new features using functions such as addition. In order to prevent violations of physical laws, domain experts have previously specified in a compatibility matrix which functions may be used for which pair of features. This is illustrated in Figure 4.5. The implementation iterates over the compatibility matrix and applies the algebraic operations or functions found there to the corresponding feature pairs. The matrix is specified with Python code and, therefore, allows a range from simple algebraic operations to complex function definitions. The newly created features are added to the existing features. In the case of a dense compatibility matrix,

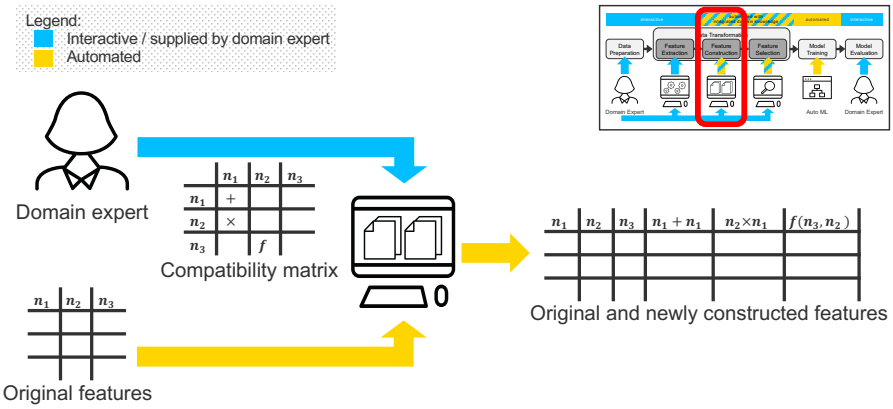

Figure 4.5: Zoomed-in view of the feature construction step of the full approach shown in Figure 4.3.

the resulting number of new features is large and is reduced again in the subsequent feature selection step.

Automated feature selection is implemented by coupling two established methods in this field with input from domain experts. Figure 4.6 is a zoomed-in view of this step. First, the implementation uses an exclude-list provided by domain experts to remove all features that are not to be used for training. Then the implementation applies a filter method from scikit-learn [PVG+11], based on an algorithm published by Pirbazari, Chakravorty, and Rong [PCR19]. This algorithm uses an estimation function to predict the influence of a feature on the output variable. It uses this information to create a ranked list of features according to their supposed importance. As part of the filter method, all features below a certain threshold of importance are then removed. The threshold value can be set automatically or by domain experts and is largely determined by the time available for training the model. With the remaining features, the implementation uses a wrapper method to determine the best set of features among those remaining. A wrapper involves training a model multiple times to evaluate the used features based on the performance of the model. Each time, it uses a different subset of features to evaluate how the inclusion or exclusion of certain features affects the performance of the model. The implementation uses a wrapper method from scikit-learn [PVG+11] with

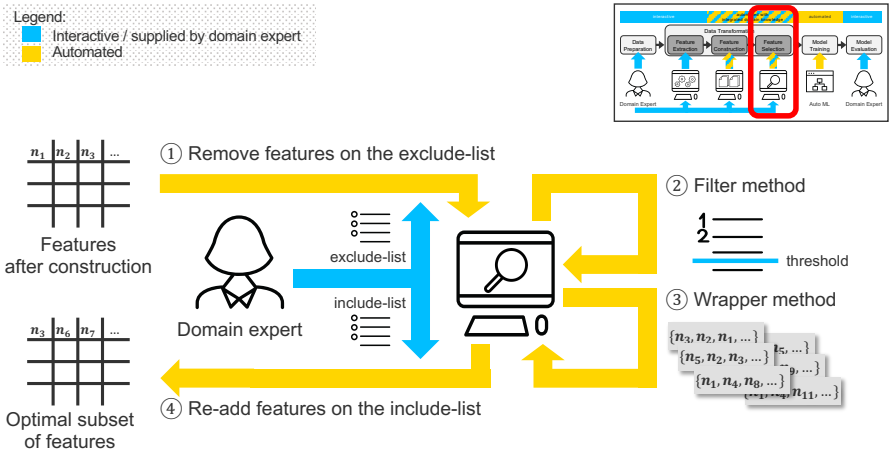

Figure 4.6: Zoomed-in view of the feature selection step of the full approach shown in Figure 4.3.

greedy forward-sequential feature selection, and evaluates the selected feature subsets based on the trained model's error. The result is an optimal subset of features. If this result misses features present on the include-list provided by domain experts, then these are now added to the subset of optimal features.

In model training, the implementation uses the previously determined optimal subset of features to train all suitable models. Since the implementation was developed specifically for the use case in Section 4.1, only models for regression analysis are supported. Based on the available raw data, the following regressors from scikit-learn [PVG+11] are supported by the implementation: ElasticNet, Ridge Regression, Kernel Ridge Regression (KRR), and Support Vector Machines (SVMs). This also includes all available kernel functions, if applicable. Following a fully automated approach for model training, the implementation performs hyperparameter optimization to individually optimize each model, as different models usually require different hyperparameters for optimal prediction. This is achieved by an exhaustive grid search for hyperparameter optimization within given parameter ranges. Here, training times are sufficiently low to allow for this exhaustive approach, which ensures the discovery of a global optimal solution rather than a local optimum. At the end

of the hyperparameter optimization, the model with the best prediction quality is selected.

Each trained model is evaluated using the R^2 score, also known as coefficient of determination. It quantifies the proportion of the variance in the dependent variable that is predictable from the independent variables. Thus, it indicates the fit between the model predictions and the actual observations. The best possible score is 1.0 and signifies perfect predictions. Scores can be negative and arbitrarily low because a model can make arbitrarily inaccurate predictions. To ensure the model's generalization capabilities, a train-test split of the data with an 80%/20% ratio as well as cross-validation techniques are utilized.

4.4.2 Case Study

This section addresses the use case presented in Section 4.1. The objective is to create a regression model based on historical simulation data. This model shall predict the collected particles for a given virtual prototype within an acceptable margin of error compared to the long-running computer simulation.

A general prerequisite for such a prediction model with low error is a suitable training dataset. In this use case, desired data was produced by a specific computer simulation, with specific applications, comparable simulation parameters, simulating a specific material. Consequently, the selection of data already requires a high degree of domain knowledge, in addition to the requirements for the fine granularity of the selection [ZRKM20; ZRKM21]. Therefore, this is best performed by domain experts. To perform such a fine-grained selection of data by domain experts, a solution such as the graph-based metadata model Graph'MeMo introduced in Chapter 3 may be used.

The following actions are taken in the respective steps of the approach presented in Section 4.3.1 and illustrated in Figure 4.3:

Data Preparation: The domain experts use tools to prepare their raw data for training by eliminating unnecessary metadata such as spreadsheet headers. They also perform domain-specific normalization of layer numbering for the layers of the filtration material. In the raw data, layers are assigned so that each prototype ends with layer 16, but shorter prototypes may begin with different layers, such as layer 4. As a result, the layers in the various prototypes are not numbered consistently. To rectify this, domain experts normalize the

data, ensuring that all prototypes start at layer 1, and shorter prototypes may end with a layer lower than 16. They also remove time steps that are not the starting time (t_0) or end time (t_{max}), which also implicitly removes the features concerning particle size from t_0. This is because, prior to the simulation's first iteration, no particles are present within the virtual prototype's filtration media. This further limits the features available for training, and thus increases the difficulty of the problem.

Since the data source is a reliable and known computer simulation, the data quality is presumed to be satisfactory, and no further investigation into data quality is conducted.

Feature Extraction: The domain experts decode a single column containing an identification of the virtual prototype, layer, and time step into three distinct features. They also generate the pressure drop gradient as a new feature, which is the derivative of the existing pressure drop. Based on the experiences of domain experts, the gradient is a useful feature to detect fast changes and patterns.

Feature Construction: In this step, domain experts are supported by automation to construct additional features, based on the existing features presented in Table 4.1 and those added in the previous step. For this purpose, the domain experts first create a compatibility matrix to define for which features the construction of new features is desired (see Section 4.3.1 and Figure 4.5). For each cell in this matrix that corresponds to a pair of features, they specify all possible mathematical functions that can be used to construct new features from this pair of features. They define this matrix in Python and the implementation loads it as input. The domain experts create the compatibility matrix based on the physical properties and units of measurement associated with the features. This prevents erroneous calculations, such as inadvertently combining length values with air pressure values, which would produce physically implausible results. The implementation then constructs all viable combinations of new features, using the functions from the matrix. The result is a list of features that combines the existing features with the newly constructed ones.

Feature Selection: Depending on the compatibility matrix created by domain experts during feature construction, the number of features may increase substantially. To address this problem, this step uses feature selection to reduce

the set of features to those that have the greatest impact on the performance of a model (see Section 4.3.1 and Figure 4.6). Training with fewer features also decreases training time, which means more time is available for optimizing the hyperparameters of a model.

In this step, domain experts are supported by automation and contribute their knowledge by creating two lists: an exclude-list and an include-list. The exclude-list is a list of features that the domain experts do not consider useful or even obstructive for training the model. The include-list contains all those features that the domain experts consider important and should always be part of the training of a model, e.g., to include certain physical relationships.

First, the implementation removes those features found on the exclude-list from the whole set of features that resulted from the previous feature construction step. The implementation then uses a filter method with this list of features to produce a ranked list of features. Based on this ranked list of features, the domain experts decide up to which rank features are considered for further processing. Now the implementation determines the optimal subset of the remaining features by utilizing a wrapper method. This results in an optimal subset of features to be used for model training in the next step. However, the implementation concludes this step by using the include-list to add features back to the optimal subset of features if they have been removed during the filter or wrapper method.

Model Training: Given the optimal subset of features from the previous step, the selection of a suitable model and the optimization of its hyperparameters can now be fully automated without the need for manual intervention. Such fully automated approaches for model selection and optimization are widely available and can achieve better results than a manual approach [HKV19]. The implementation uses the subset of features to train all available regression models and iteratively optimize their hyperparameters until each model reaches its optimum. However, the best model is not selected automatically. Instead, the results are evaluated by domain experts in the next step.

Model Evaluation: Evaluating the trained models is automated with an interactive verification by domain experts. The implementation pre-selects and ranks trained models based on their R^2 score in cross-validation during the model training step. Note that cross-validation is performed on training data

and not test data to prevent leaking knowledge about the test data into the model. The most promising models and the subset of features they are trained with are visualized and presented to domain experts that can then reason about the results. These can be seen in Table 4.2 in the following Section 4.4.3.

4.4.3 Evaluation and Discussion

The implementation was run on a virtual machine running Ubuntu 18.04 LTS, equipped with 16 CPU cores and 128 GB of main memory. This virtual machine is hosted on a private cloud-computing platform built with OpenStack by the Institute for Parallel and Distributed Systems (IPVS) at the University of Stuttgart. The underlying hardware consists of a cluster powered by Intel®Xeon®E5-2683 v3 processors. These hardware resources were not overprovisioned. A full run of this scientific setup, designed for optimal results and the collection of extensive data for the purpose of this evaluation, takes a little less than three days. For real-world applications, a more intelligent scheme with a non-exhaustive approach may be adopted to save time. For example, instead of an exhaustive grid search, a different optimization algorithm may be used that saves time but may miss a global optimum.

To establish a baseline for comparison, the evaluation additionally used the original set of prepared and extracted features from the raw data. This set of features bypassed feature construction and feature selection, while still passing through all other steps. So, automated machine learning, including model selection and hyperparameter optimization, was still carried out in this baseline.

The initial evaluation revealed significant fluctuations in the standard deviation of R^2 scores from models obtained through optimization with 5-fold cross-validation. The error of prediction models demonstrated similar inconsistency, exhibiting very low error for certain virtual prototypes and very high error for others. This may suggest a potential overfitting issue. To address this, the optimization changed to a 10-fold cross-validation and rerun all parts of the evaluation. This adjustment proved successful in reducing standard deviations. Moreover, the uniformity of prediction error improved noticeably, and the standard deviation decreased.

Table 4.2: Comparison of the best model of the baseline and the full approach.

	Baseline	Full Approach
Algorithm	KRR	KRR
Kernel	Linear	RBF
R^2	0.4880	0.8281
SD	0.3509	0.1866

All four regression algorithms listed in Section 4.4.1 were evaluated for both the baseline and the full approach, which additionally includes feature construction and feature selection. Table 4.2 summarizes the results for those regression algorithms that performed best for the baseline and for the full approach, respectively. In both cases, this is a KRR model with a different kernel. The table also shows results of two evaluation metrics of the models: the R^2 score evaluated on test data and the standard deviation (SD) of this R^2 score for each fold of the 10-fold cross-validation on training data.

For the baseline, a trained model with KRR and a linear kernel achieved the highest R^2 score of 0.4880 on test data. The standard deviation of the R^2 scores for each fold of the cross-validation during training of this model was high at 0.3509. Considering this low R^2 score, the utility of such a surrogate model is limited, a sentiment echoed by domain experts. They further raised concerns about the high standard deviation of 0.3509 during cross-validation, which undermines the reliability of this model's predictions. The residual plot for this result exhibits an open funnel pattern, a characteristic that hints at potential enhancement opportunities through feature engineering [TCW98].

In the case of results achieved with the full approach, i.e., with all steps of the machine learning pipeline and its feature engineering, SVMs consistently performed worse than KRR. The trained model with the overall highest R^2 score included constructed features during training and used KRR with an Radial Basis Function (RBF) kernel. It achieved an R^2 score of 0.8281 on test data and a standard deviation of R^2 scores across the cross-validation of 0.1866 on training data. This is also one of the lowest standard deviations among the trained models and roughly half the standard deviation of the best baseline model.

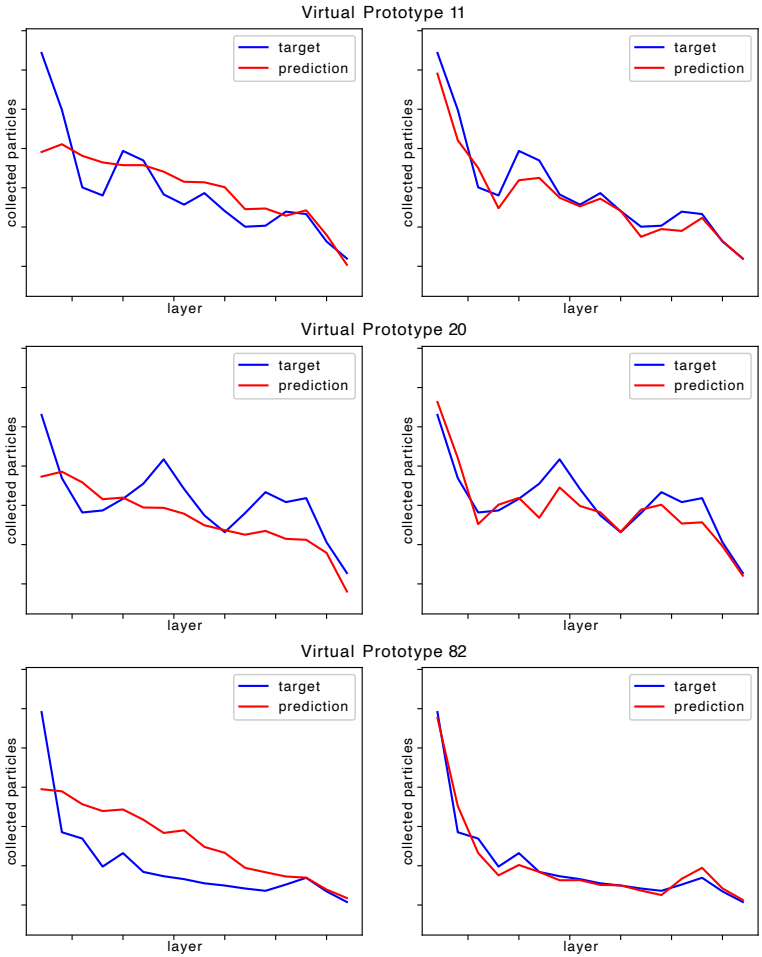

Figure 4.7: Predictions made by the baseline model (left) compared to those made by the model from this case study (right).

An intuitive understanding of the implications of these R^2 scores is difficult, and domain experts need support to verify if a prediction is sufficiently accurate for a given task. Here, a visualization of the target value and its prediction is an effective approach. To illustrate the improvement in R^2, Figure 4.7 displays the actual target values alongside the predictions made by the trained models. Figure 4.7 shows these values for selected virtual prototypes with representative

average improvement. The left graphs are predictions by the baseline model, the right graphs are predictions by the model that resulted from this case study. The y-axis represents the actual simulation value for collected particles in blue and the predicted value in red. The x-axis corresponds to the layer, signifying the depth within the filtration media of the virtual prototype. Essentially, this visualization demonstrates the accumulation of particles at different depths in the virtual prototype, providing valuable insights and metrics for optimizing filtration media. Here, the visualization shows a clear improvement of the prediction quality due to the feature construction and selection with integrated domain knowledge. The prediction is significantly improved, and domain experts can now assess whether this prediction is suitable for a use case.

These results demonstrate that the proposed approach effectively addresses both the data and knowledge challenge (see Section 4.1.4) in the development of data-driven surrogate models for simulations in virtual product development. The approach addresses the challenges as follows:

C_D Data Challenge: Despite the limited availability of simulated prototypes due to time-consuming simulations and the inability to mix data from different materials or manufacturing processes, the proposed approach effectively leveraged the available data. By using advanced feature engineering techniques, the approach was able to significantly improve the prediction quality, which was low due to the limited data and features. The approach extracted the most relevant information from the data, ultimately leading to a more accurate and trustworthy surrogate model. Further, the data selections could be performed by domain experts using Graph'MeMo, which provided the necessary specific datasets for training surrogate models. The success of this approach in handling the data challenge is evidenced by the significant improvement in R^2 scores, indicating a more reliable prediction of collected particles for the filter media.

C_K Knowledge Challenge: The lack of domain expertise in feature engineering was addressed by integrating domain experts and their knowledge through interactive means into the creation of a data-driven machine learning pipeline. Domain experts provide valuable insights on the science and physical properties used in the features, ensuring that the

constructed features made physical sense and did not introduce errors or unreliability. Further, the possibility to influence feature selection allows domain experts to guarantee training on features that are known to be relevant in the physical system that is to be modeled. This ties the trained model closer to laws of physics and increases reliability for domain experts. Through the exclusion of features, domain experts prevent training a model based on knowingly irrelevant data and positively influence training and optimization times. This approach proved essential to the development of a successful data-driven surrogate model, as evidenced by the substantial increase in R^2 scores when incorporating domain knowledge into the feature construction and selection.

In conclusion, the case study demonstrates the effectiveness of the proposed approach in overcoming the data and knowledge challenges inherent in the development of data-driven surrogate models for long-running computer simulations. By leveraging advanced feature engineering techniques and integrating domain experts into the process, the approach was able to achieve improvements in prediction error and model reliability, ultimately leading to more effective optimization of filtration media.

4.5 Summary

This chapter describes a data-driven surrogate modeling approach for long-running computer simulations in virtual product development. The primary challenges are the limited availability of simulation data and the need to incorporate domain knowledge into the feature engineering process. The proposed solution involves a systematic approach to data preparation, feature extraction, feature construction, feature selection, and model training. By leveraging advanced feature engineering techniques and integrating domain experts into the process, the approach effectively addresses the data and knowledge challenges. The approach creates a physically sound machine learning pipeline with integrated domain knowledge that can, subsequently, be automatically updated and improved in a data-driven fashion.

The success of the proposed approach is evidenced by the improvement of prediction error and reduced variance, indicating a more accurate and

trustworthy surrogate model. The case study and evaluation demonstrates the feasibility of data-driven surrogate modeling in this engineering domain, and highlights the importance of integrating domain expertise and advanced machine learning techniques for the successful development and deployment of such models.

Chapter 5

Graph-based Process Discovery in Virtual Product Development

Organizations need knowledge about their processes in order to improve and optimize them. Process mining is an umbrella term for the discovery and analysis of real-world processes and is increasingly employed in various application domains [dSMF+19]. In the manufacturing industry, e.g., it can mine operational knowledge from event data of machines and sensors and use it to identify and optimize issues in performance and compliance of manufacturing processes [DRG20]. Process discovery is a typical first step in process mining to derive a previously unknown process model [van+12].

This chapter proposes a novel approach to process discovery that is applicable and tailored to product development projects. First, this approach uses Graph-based Metadata Model (Graph'MeMo) and its graph-based data structure that is able to represent all activities and data of product development projects instead of event logs (see Chapter 3). Graphs are capable of representing the provenance information available in Product-Lifecycle-Management (PLM)

systems [ABD+14]. Graph'MeMo can retain the provenance information about activities and their dataflow, and that can then be used to discover process models for development projects. Therefore, an otherwise lossy conversion of data from PLM systems to the event log format is not necessary.

Second, this chapter discusses how this graph structure facilitates searching for frequent patterns. Single and common process models cannot adequately describe the different variants of highly unstructured development projects. Patterns are frequently occurring sequences of activities that accurately describe a subset of a larger process. Instead of a single convoluted and approximated process model, patterns allow to describe each variant of unstructured processes as a set of accurate and precise patterns. This chapter discusses Frequent Subgraph Mining (FSM) algorithms to discover patterns in the graph structure representing product development projects and describes the application of these algorithms step by step. It evaluates this approach with a prototype and with generated data that represents common product development projects. This determines whether the approach can discover all relevant patterns in the data, and how well FSM algorithms can handle errors and noise in the data. Based on an example, this chapter discusses how the identified patterns may help engineers to improve the execution of their product development projects.

This chapter presents contribution C_3 and addresses research challenge RC_3. The contents of this chapter are an extended and revised version of an earlier publication by the author [ZRS+22].

The remainder of this chapter is structured as follows: Key characteristics of product development projects are described in Section 5.1. Section 5.2 discusses why related approaches and algorithms to process discovery are not suitable to address these key characteristics. Section 5.3 describes the graph structure and the approach of using FSM. Section 5.4 describes the method and results of the evaluation. Section 5.5 concludes and lists future work.

5.1 Motivation and Key Characteristics of Product Development Projects

Analyzing and improving the product development process can be of great benefit to an organization. The time it takes to develop a product affects both

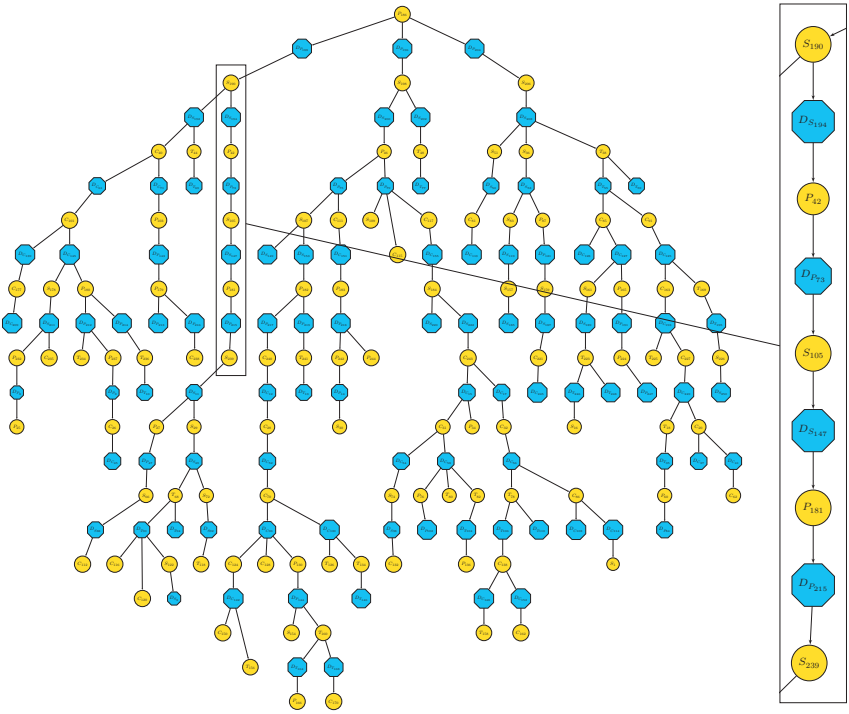

Figure 5.1: Instance of a product development project represented by the proposed graph structure. Activities are yellow nodes, data are blue nodes. Edges indicate dataflow. The magnified box at the right shows a pattern of a long sequential sub-process of this development project [ZRS+22].

the cost and the chance of winning the contract for a product. To achieve this, an organization needs knowledge about their processes and the first step is to discover process models. These can then subsequently be used for analyzing and improving these processes.

As an example, the execution of a product development project may look as depicted in Figure 5.1. This illustration shows a development project as a graph. Yellow nodes represent work activities, blue nodes represent data sets, and edges connect nodes to indicate dataflow. On the surface, the illustration exhibits a lot of parallel activities, which is desirable to reduce the

Table 5.1: Key characteristics of product development projects (see Section 5.1) and assumptions of process discovery algorithms that are incompatible with these key characteristics (see Section 5.2) [ZRS+22].

Product Development Key Characteristics	**Assumptions of Process Discovery Algorithms**
Project data and provenance information stored in PLM systems.	Machine-generated and reliable event logs as input data.
Unstructured processes, e.g., due to human decision-making.	Existence of one single or generic process model.

time needed for product development. However, as the magnifying box on the right of Figure 5.1 reveals, there are also long sequences of sequential activities that may, however, also be performed in parallel. The superficially parallel appearance masks these sequential work patterns, which are a subject of interest for optimization. Patterns can be identified and classified as either beneficial or detrimental regarding certain metrics of product development, such as development cost or time. The use of beneficial patterns, such as cost-effective computer simulations, can be audited or promoted. Detrimental patterns, such as extensive cost-intensive prototyping, can be discouraged. Process discovery may help to analyze these actual process executions, and, thus, help to identify such problems and optimize projects. However, two key characteristics of product development projects prevent the usage of existing approaches to process discovery, as summarized in Table 5.1 and detailed below.

5.1.1 PLM Systems

Data generated and decisions made during product development are commonly stored in information systems, such as PLM systems. PLM systems contain all information of a product at any given state in its lifecycle, and, thus, also for the early stage of product development [HDGL13]. They even contain provenance information about the activities performed in projects [GT08]. In the context of product development, this may refer to information on how a simulation result was derived (input data and simulation parameters) or to information

for which product design (2D or 3D model) a prototype was built. In other words, provenance information of PLM systems mainly describes the relations between activities and the data these activities produce or consume. Hence, it describes the detailed structure and dataflow of the processes involved in individual development projects. Furthermore, provenance is vital to improve collaboration, general understanding, and reproducibility of processes [GT08].

5.1.2 Unstructured Processes

Product development projects are unstructured processes due to a large amount of human interference and decision-making, which is often done by a team of engineers [van11b]. All projects are comprised of a set of abstract steps ranging from product planning, design (Computer-aided Design (CAD)) over computer simulation (Computer-aided Engineering (CAE)) to physical testing (Computer-aided Testing (CAT)). However, the detailed control flow of individual projects within and between these abstract steps greatly varies. The development process is steered by the experience and intuition of engineers mixed with trial-and-error characteristics. Engineers may explore multiple prototypes in parallel or in sequence. After each computer simulation or physical product test, it may be necessary to go back to the drawing board and iterate on the prototype [ZRKM21]. This is usually necessary when a virtual or physical test indicates that the performance characteristics defined in the product specification cannot be achieved or are even unnecessarily far exceeded. Prototypes also may be discarded completely at different states of the process and for different reasons. For instance, the development costs may reach a certain threshold or an intended material may become unavailable in the necessary quantities.

Not only the control flow between activities but also the activities themselves may vary. For example, engineers can create a virtual prototype using different CAD applications, depending on personal taste, their experience with an application, or its suitability for the given task. Activities also produce and consume large amounts of data, such as the results of a computer simulation. These results may be reused in the same development project or even in other projects to replace another development activity. For example, previous simulation

results may be reused to avoid a time-intensive computer simulation (CAE) activity in subsequent projects.

Modern decision-making systems may support engineers and result in more structured processes. However, such systems are based on process knowledge that is usually not available a priori [dSMF+19]. In addition, the lack of stringent structure in product development projects is often even intentional. Innovative and creative products have no universal blueprint and require leeway in their development projects. In fact, strict guidelines and adherence to automated decision-making may hinder innovation. There are many influencing factors that contribute to many decisions. Some of them even change over the course of the project, e.g., requirements of external customers.

5.2 Related Work

Process discovery algorithms take event logs as input and reconstruct a process model that adheres to the sequence of the logged events [van11c; van12]. An event log is a collection of recordings of individual events, also called *activities*. Activities symbolize a task in a process, e.g., gluing a workpiece. Each activity is associated with a timestamp. A sequence of activities sorted over these timestamps is called a *trace*, which represents the recording of a whole process execution. The event log for a process is a collection of all its traces. Logs of activities are usually generated by multiple machines used for manufacturing. These systems or machines either commit individual records of each activity to the log, or write individual logs that are later merged. So, an event log does not contain any explicit dependencies between several activities. Instead, process discovery algorithms attempt to reconstruct this information based primarily on the timestamps of activities and the frequency of each trace.

The assumption of having machine-generated and reliable event logs as input data (see Table 5.1) is incorrect for the domain of product development [Gv07]. Instead, product development uses PLM systems to store their data and provenance about activities performed in development projects [GT08]. To apply classic process discovery algorithms, analysts first need to convert PLM data into the event log format. However, this conversion loses the valuable provenance information contained in PLM systems because event logs cannot represent

this kind of provenance. So, it loses information about the dataflow between activities, i.e., information about dependencies between activities and the data they produce or consume.

In general, event logs only offer a view from a specific angle on a process and, thus, an incomplete picture [van11c]. There are attempts to use more information for process discovery than available in a classic event log. Leemans, Fahland, and van der Aalst add lifecycle information in form of start and end timestamps for activities, i.e., each activity then has two timestamps instead of only one [LFv16]. They argue that process discovery algorithms have difficulties detecting parallelism because activities in event logs are treated as atomic operations. Their improvement helps distinguish between real parallelism and concurrency (quasi-parallelism) of activities in traces. However, such extensions do not represent data and explicit dataflow dependencies. Thus, they are far from being able to take into account all provenance, which is usually available in PLM systems.

Günther and van der Aalst performed case studies in various companies [Gv07]. They found the assumption of one single and generic process model (see Table 5.1) to be incorrect for less-structured processes, as often found in real-life environments. Van der Aalst describes less-structured and unstructured processes as those where trial-and-error or the experience and intuition of humans drive activities [van11b]. Structured processes, on the other hand, are those whose activities are repeatable and where most activities can be automated. Classic process discovery is not well suited for product development, as this domain comprises highly unstructured processes. Applying process discovery to these unstructured processes leads to so-called *spaghetti* process models [van11b]. These are huge and convoluted models that are unmanageable and difficult to understand. In the context of product development, they are especially not able to adequately describe all individual variants of development projects, which then finally cannot be analyzed anymore.

A general issue in the field of process discovery, which affects all algorithms working with event logs, is representational bias, as argued by van der Aalst [van11a]. This means that different algorithms may discover substantially different process models from the same event log due to different assumptions that each algorithm makes. Van der Aalst further defines four di-

mensions of quality for discovered process models [van11a]. Leemans, Fahland, and van der Aalst argue it is not possible to find a model that scores equally well on all dimensions [LFv13]. For instance, the α algorithm [vWM04] is among the most prominent basic algorithms for generating process models based on event logs [vWM04]. It assesses several relations between activities based on their occurrences in traces of event logs. An example of these relations is direct succession of activities. For two activities a and b, it assumes a direct succession from activity a to b ($a > b$) if and only if there exists a trace where a is directly followed by b. Other relations consider parallelism, ($a||b$), which exists if and only if $a > b \wedge b > a$. Further relations are causality and choices. Alves De Medeiros et al. created $\alpha+$, an extension of the α algorithm that allows reliably detecting short loops of length 2 and 1 [AvvW04]. They accomplish this by wrapping the original α algorithm with a pre- and post-processing step. Pre-processing detects short loops and removes them from the event log, which is then processed by the α algorithm. Their post-processing step then inserts these previously removed loops into the process model generated by α. Wen et al. created another extension, called $\alpha++$, to correctly detect non-free-choice constructs [WvWS07]. $\alpha++$ adds further relations to α and modifies the algorithm itself instead of wrapping it. It allows the reliable discovery of implicit dependencies between activities, commonly found in many real-life processes where control flow is based on human decision-making. However, all these approaches are susceptible to noise and incompleteness in event logs. This refers to erroneous activities which are either redundant, not contained in the log, or carry incorrect data, e.g., a wrong timestamp. This lets the algorithms assume incorrect relations, and it results in needlessly complicated or incorrect process models. Further disadvantages include difficulties in detecting loops and cascading merges [van11c].

Over time, many authors contributed more sophisticated algorithms for process discovery. Although they are still all based on event logs, the different approaches of these algorithms can be grouped into categories.

Genetic miners start with an initial model and iteratively mutate it, assess its fitness with an evaluation function, and select the fittest models for the next iteration [van11c]. The resulting process models of these approaches tend to overfit, are susceptible to noise, and require high computational power.

Heuristic approaches to process discovery offer greater resilience against noise and incompleteness of event logs [van11c]. Such algorithms mainly employ the frequency of activities, i.e., their number of occurrences in traces, to determine which activities are to be considered for a process model. This essentially filters out all rare activities, as it assumes them to be noise and not important. However, as the processes in product development are unstructured and rich in variance, some rare, but still valid activities may exist and be of interest. These rare, but valid activities are then wrongly assumed as noise by heuristic approaches and may, thus, be missing in the final process model.

A similar strategy is used by Günther and van der Aalst, who aim to prevent spaghetti models with their *Fuzzy Mining* approach [Gv07]. Instead of relying on frequency, they define various metrics for different aspects of a process model, e.g., significance and correlation of activities. These metrics tune the parameters of their discovery algorithm, which operates on a pre-filtered event log and constructs a large graph. To derive a process model, they prune this graph by removing and aggregating nodes and edges. However, finding a correct process model requires carefully tuned parameters each time process discovery is carried out. This is time-consuming and complex and, thus, not feasible for engineers in product development.

Leemans, Fahland, and van der Aalst propose an inductive miner to discover what they call an 80% model [LFv13]. Applying the Pareto principle, the authors claim that only 20% of a theoretically complete process model are required to explain 80% of the observed behavior. Their algorithm uses a divide-and-conquer approach to recursively split the activities of the event log and filter infrequent activities and traces. Splits are performed when the algorithm recognizes certain behavior that is similar to the relations of the α algorithm, e.g., parallelism. They use local filtering of infrequent behavior because it performs better than global filtering prior to process discovery in metrics such as precision and simplicity of the model. This approach performs better than most others at discovering an 80% model, but it often yields a worse precision of the model.

Maggi, Bose, and van der Aalst take a different direction and search for declarative process models [MBvdA12]. These are based on parameterized constraint templates that explicitly express the valid relationships between

activities. They construct a large set of all the possible constraints among all activities and then use various metrics to reduce this large set, e.g., the number of traces where a constraint is true. This approach produces one single process model and is also based on event logs and, therefore, utilizes no provenance information.

5.3 A Graph Structure to Discover Process Patterns

This section proposes to use a graph structure as an alternative data structure to discover process patterns. A graph structure is capable of representing the provenance contained in modern PLM systems [ABD+14; DF08]. Graphs offer explicit modeling constructs to not only describe the performed activities within a project, but also the data connected to these activities as well as relations between data and activities [ZRKM21]. Such relations may essentially describe the dataflow, e.g., which data were produced and consumed by which activities.

The graph-based metadata model Graph'MeMo introduced in Chapter 3 provides all these properties. It can represent the data sets as well as activities in product development projects. Its relationships between data and activities model the dataflow. Thus, it can directly serve as a suitable data structure that represents the rich information contained in modern PLM systems.

In addition, a graph structure may be composed of a set of subgraphs. To this end, subgraphs may represent frequent patterns that accurately describe subsets of the large processes of product development projects. In contrast to a convoluted spaghetti process model [van11b], these patterns allow for describing each variant of these unstructured processes in a flexible way. Analyses based on these patterns can yield best practices or uncover deviations from organizational guidelines for executing development projects. A common way to identify patterns in graph structures are FSM algorithms, such as GraMi or gSpan [EASK14; YH02]. These algorithms search for commonly occurring subgraphs in a set of graphs.

Section 5.3.1 introduces a reduced form of Graph'MeMo as graph-based data structure. Section 5.3.2 discusses how to use FSM to identify process patterns from a set of graphs as well as which kinds of FSM algorithms are suitable for

this task. Section 5.3.3 describes the whole approach of using FSM for process discovery step by step.

5.3.1 Graph-based Data Structure

The data sources in product development are PLM systems, which are an integral part of the information systems found in manufacturing companies. PLM systems contain all information of a product at any given state in its lifecycle, and, thus, also for the stage of product development [HDGL13]. Their data also includes provenance information about the activities performed in product development [GT08]. Provenance is information that describes the process by which a result was obtained [HDB17]. There exist different categories of provenance. In this context of product development, provenance refers to information on how an activity was performed. For example, how a computer simulation result was derived (input data and parameters), or based on which product design a prototype was built. In other words, provenance information describe the relations between activities and the data these activities produce or consume. This corresponds to coarse-grained workflow provenance, which is known to be an important part of efficient collaboration and general understanding and reproducibility of processes [HDB17].

A transformation of these data to event logs is not possible without losing the provenance because event logs cannot represent this category of provenance. To prevent such a loss of information, this section proposes a data structure capable of representing the whole processes of product development projects, i.e., their activities, data, and the relationship between these. Explicitly considering data and their relationships to activities is important because it provides an additional source of information about development projects, i.e., their dataflow. It can indicate if human decisions were derived based on data, or it can serve as an indicator for substituted activities. For instance, it may indicate when data of an activity has been reused in a project. For example, during the progression of a development project, previously computed data may be reused multiple times instead of repeating the performed activity.

A data structure capable of representing activities, data, and their relationships is a labeled graph. A labeled graph, as needed here, can be defined as $G = (V, E, L_V)$ and consists of a set of nodes V, a set of edges $E \subseteq V \times V$,

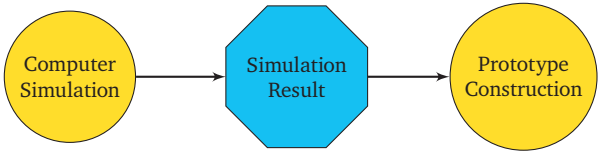

Figure 5.2: Exemplary graph structure for a small part of a product development project. Activities are yellow nodes, data are blue nodes. Edges indicate dataflow [ZRS+22].

and a labeling function L that assigns labels to nodes. This structure is capable enough to represent a product development project by mapping project activities and their data to nodes [ZRKM21]. This is illustrated in Figure 5.2, where each activity and each data set is abstracted by a node. A node is associated with a label that identifies the specific activity or data set. In Figure 5.2, the labels of activities (yellow nodes) indicate the execution of a computer simulation or the construction of a prototype for product testing. The label of the data set (blue node) characterizes this data as result of the computer simulation.

The logical metadata model of Graph'MeMo is such a graph structure and capable of representing activities and data sets as nodes (see Section 3.3). It also associates activity and data nodes based on the dataflow as it occurs in a product development project. For the application here, a subset of Graph'MeMo's graph structure is sufficient. Graph'MeMo is also able to attribute activities and data with metadata through an additional node type in its graph structure. However, this node type for metadata is not necessary for the application here and is therefore omitted.

Edges connect those activity and data nodes that share a relationship. For this approach to process discovery, the most relevant data are dataflow dependencies that are part of the provenance information contained in PLM systems [ABD+14]. So, a directed edge connects activity and data nodes with each other. The direction of an edge represents the direction of the dataflow. Each activity node may have edges pointing towards an arbitrary number of data nodes that represent the output data produced by this activity. An activity node also may have incoming edges from arbitrary data nodes, representing which data this activity consumes as input. In Figure 5.2, the computer sim-

ulation activity produces simulation results that are then consumed by the prototype construction activity.

An edge usually does not directly connect two activity nodes with no data node in-between. Doing otherwise would imply that two activities have a causal relationship with zero transfer of information between them, i.e., no dataflow at all. Such scenarios can be constructed in theory, but are rare in practice because activities without dataflow are usually independent of each other and then not connected by an edge.

Altogether, the whole graph then covers the process executions of a product development project. It covers all development activities carried out by engineers, the data these activities produce or consume, and an explicit description of the dataflow. This representation as a labeled graph is hence sufficient for the approach to process discovery proposed here.

Note that *Directly-Follows Graphs* [van19], frequently used in process mining, are no suitable alternative. They are constructed based on event logs and contain the same level of information. They model activities as nodes and connect these according to their *directly follows* relationship, i.e., a heuristic also used in the α algorithm [vWM04]. So, they cover a potentially incomplete control flow between activities, but they are not able to describe involved data or the dataflow in product development projects. Hence, Graph'MeMo's graph structure is superior in terms of its expressive power to conduct process analysis.

The previously shown Figure 5.1 depicts an instance of a graph structure that is based on the abovementioned graph model, and represents a large and unstructured product development project with all its activities and data sets. At first glance, this unstructured process appears to include many parallel activities. Nonetheless, when employees tasked with process optimization zoom in to take a closer look, they can discover several sequential work patterns, e.g., as shown in the magnifying box on the right. Here, several iterations of product planning and computer simulation take place sequentially, although they may be performed in parallel. However, an organization may want to parallelize as many activities as possible in development projects to reduce time to completion. It needs to discover the process models of this actual project execution in order to identify and optimize such problems.

Unfortunately, existing process discovery algorithms generate a single and convoluted process model for such unstructured processes, i.e., a so-called spaghetti model [Gv07; van11b]. Such spaghetti models are difficult to understand. In particular, it is hard to identify all details of individual process variations, as they are present in unstructured processes. So, instead, the idea is to search for patterns that describe these variations of unstructured development projects. Employees tasked with process optimization may discover many frequent sequential work patterns during the execution of projects, as it is the case in the process shown in Figure 5.1. Subsequently, they can analyze why this is the case and optimize their project execution towards a more parallel approach. This graph structure allows them to discover such patterns because it includes all activities and data of development projects and explicitly represents their dataflow.

Discovering patterns in the processes of product development projects may also be beneficial in other ways. Frequently occurring patterns in successful projects may provide knowledge about valid (sub-)processes. Used as building blocks or *best practices*, inexperienced engineers can exploit this knowledge for their decision-making. Furthermore, an organization can compare discovered patterns as they occur in the real world with predefined guidelines for development projects. A deviation indicates opportunities for improvement and potential for optimization in project execution. The following subsection discusses how to use frequent subgraph mining to discover process patterns in these graph-based structures.

5.3.2 Frequent Subgraph Mining to Discover Process Patterns

With a graph-based representation of product development projects, FSM can discover process patterns in these projects. For the approach presented here, these patterns are frequent subgraphs of the whole graphs of development projects. Given a graph $G = (V, E, L)$ as introduced in Section 5.3.1, a *subgraph* $G_S = (V_S, E_S, L_S)$ is a subgraph of graph G if and only if $V_S \subseteq V$, $E_S \subseteq E$, and $\forall v \in V_S : L_S(v) = L(v)$ [JCZ13].

A major difference between individual FSM algorithms is their desired input. Some algorithms require a list of smaller graphs and search for subgraphs that are shared across the given input graphs. Other algorithms operate on one

large graph and search for subgraphs exclusively within this single graph. The approach presented here supports both to maintain compatibility with the large amount of FSM algorithms (see Section 5.3.3). Algorithms also differ in their definition of a valid input graph, e.g., whether edges are directed or undirected, or whether cycles are allowed [JCZ13]. Cycles and duplicated edges are frequently not allowed [JCZ13]. Suitable algorithms for the approach presented here have to support directed edges because they are needed to explicitly indicate the direction of dataflow in processes.

An important parameter of FSM algorithms is the so-called *minimum support*, which corresponds to the frequency of specific subgraphs in the input graph(s). The frequency of subgraphs is determined by their number of occurrences. Depending on the specific algorithm, this may be the actual number of occurrences in one large graph or the number of input graphs in which this subgraph is found at least once. This means that a subgraph with high support is present in many graphs of the input data. So, it occurs frequently in many product development projects and may, hence, be seen as a common process pattern of these projects. Therefore, the minimum support of FSM algorithms is a suitable threshold to indicate how many times a subgraph has to occur in the input graphs to be considered as a valid process pattern. An example is illustrated in Figure 5.3. It shows three graphs g_1, g_2, g_3 and the result of an FSM algorithm searching for subgraphs with at least a minimum support of 2. The result is a subgraph with three nodes that occurs in g_1 and g_2, but not in g_3, so the subgraph has a support of 2.

There are differences in what FSM algorithms consider a subgraph. Although a subgraph is a structural pattern, literature often explicitly refers to subgraphs when referring to connected graphs. In some use cases, a more dynamic definition of subgraphs is required and literature frequently refers to these as patterns. While a traditional approach such as gSpan [YH02] adheres to the basic definition of frequent subgraphs, other algorithms such as GraMi [EASK14] offer more flexibility. GraMi optionally regards several subgraphs that only show minor differences, e.g., a single missing edge between two nodes, as belonging to the same pattern. Applied to product development projects, such a more relaxed definition of patterns improves results when dealing with uncertainty, noise, and incompleteness in the data. It is also beneficial in case

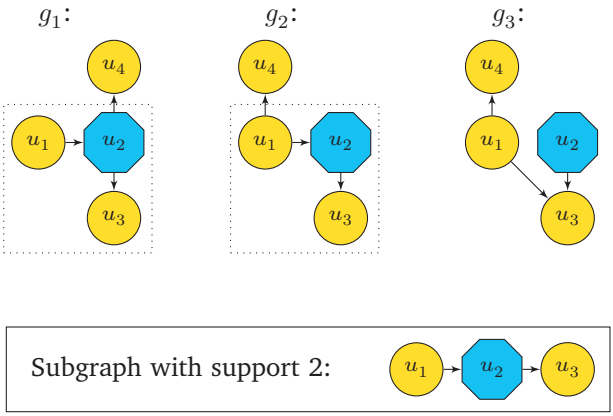

Figure 5.3: Three graphs g_1, g_2, g_3 and a subgraph with support 2 embedded in g_1 and g_2 [ZRS+22].

minor differences in development projects are to be expected due to human decision-making. Therefore, for use in product development projects, FSM algorithms that are explicitly designed to discover such more flexible frequent subgraphs are recommended [CYZH07; EASK14; PIS11; ZLGZ09].

The basic problem FSM algorithms have to solve is the subgraph isomorphism problem. Given two graphs G and H, the algorithms need to check whether there is a subgraph $G_S \subseteq G$ that is isomorphic (i.e., a bijection exists) to H. Subgraph isomorphism is known to be NP-complete [JCZ13]. However, many optimized or approximate algorithms exist that make FSM applicable to large graphs. An example is the gSpan algorithm, which drastically reduces the number of subgraphs on which isomorphism is to be tested, reducing memory consumption and allowing large graphs to be processed [YH02]. GraMi features an optional approximation mode, where it may miss some frequent subgraphs, but speeds up computation [EASK14]. If the graph structure created for product development projects can be limited to a tree, then the problem turns into searching subtrees and the linearity of the subtree isomorphism problem greatly accelerates this. There exist algorithms that can handle both. For example, GASTON assumes that the candidates for which it needs to verify isomorphism are trees [NK05]. However, the algorithm gracefully falls back to the subgraph isomorphism problem when this assumption is violated. Thus,

the search for patterns for all project graphs that are actually trees is greatly accelerated, while such projects that are not trees are still supported.

5.3.3 Approach Step by Step

The approach to discover process patterns in product development projects is an algorithm separated into four steps.

First Step: Graph Generation

For an organization that manages their data with Graph'MeMo, the necessary graph data structure to discover process patterns already exists. Merely a view on the graph structure needs to be created, which hides the metadata nodes, since they are not needed here. Only the activity and data nodes and their association is necessary for the following steps.

Other organizations have to generate the necessary graph structure. An organization typically stores its product development data in an information system that provides PLM features. Such PLM systems are usually commercial offerings from specialized vendors. These software products have different characteristics to be able to compete in the market. However, their core functionality is identical, and they all have support for sophisticated data export through query interfaces [ABD+14]. On top of this, customer-specific expansion of functionality is also widely supported. However, due to the large differences between the available PLM systems, the description of the data extraction from these systems remains at a high level of abstraction.

The first part of the proposed algorithm creates the necessary graph structure for all product development projects and is outlined in Algorithm 5.1. Based on extracted PLM data, the algorithm has to traverse the project data from the PLM system. The algorithm initializes an empty graph for each project and then traverses the data of a project. For each activity encountered in the project data, it creates a node in its graph that represents the execution of this activity. Likewise, it creates a node for each data set. It connects activity and data nodes with a directed edge when it can deduct a dataflow relationship between them through provenance information or other hints in the PLM system. For example, when the PLM system contains a request for a computer simulation

and lists data files as simulation results, it can deduct that a simulation activity took place and produced the listed data. Thus, it connects the activity node representing the computer simulation with the data node corresponding to the simulation results. The graph indicates the dataflow from the activity to the data node with the direction of the edge.

Algorithm 5.1 Discover process patterns in product development projects. First step: initial graph generation.

$graphs \leftarrow \emptyset$
$project\_data \leftarrow \emptyset$
$project\_data \leftarrow Export\_From\_PLM()$
for all $p \in project\_data$ **do**
  $V_{Data} \leftarrow \{Node(v) : \forall v \in project\_data$ where v is a dataset$\}$
  $L_{Data} \leftarrow \{Label(v) : \forall v \in project\_data$ where v is a dataset$\}$
  $V_{Activity} \leftarrow \{Node(v) : \forall v \in project\_data$ where v is an activity$\}$
  $L_{Activity} \leftarrow \{Label(v) : \forall v \in project\_data$ where v is an activity$\}$
  $E \leftarrow \{(u, v) : u, v \in V_{Data} \cup V_{Activity}$ with dataflow from u to $v\}$
  $graphs \leftarrow graphs \cup (V_{Data} \cup V_{Activity}, E, L_{Data} \cup L_{Activity})$
end for

As PLM systems in product development usually contain sufficient provenance data [GT08], the result of this step is one graph per development project. However, data may be absent due to human error, e.g., information was not correctly added to the system. In such cases, the algorithm can use timestamps of work activities and use this information to connect parts of the graph that are unconnected, based on chronological ordering. So, there are no activity or data nodes that are unconnected.

Second Step: Pre-processing

There needs to exist a root node for each project graph that can reach all other nodes in this project graph, disregarding the direction of edges. The pre-processing step of the proposed algorithm ensures that such a node exists and adds one if this is not the case, as outlined in Algorithm 5.2.

This is necessary for some FSM algorithms and also a logical assumption for development projects. Each activity in product development needs some amount of information as input. To design a product (CAD), it is necessary to

know which criteria and characteristics the product has to match. To simulate (CAE) or test (CAT) a product, it is necessary to know which product to test and which characteristics to simulate. The only starting point without input data or preceding activities is the initial specification that triggers a product development project. If such a node does not exist, e.g., because a record is missing from the PLM, the proposed algorithm creates a new node that represents the start of the project. This node is then connected to all other nodes of the graph with no incoming edge.

Algorithm 5.2 Discover process patterns in product development projects. Second step: pre-processing.

for all $g \in graphs$ **do**
  if $\neg Has\_Root(g)$ **then**
    $(V, E, L_V) \leftarrow g$
    $V \leftarrow V \cup Make\_Node(root)$
    $L_V \leftarrow L_V \cup Make\_Label(root)$
    $U \leftarrow \{v \in V : \forall w \in V : (w, v) \notin E\}$
    $E \leftarrow E \cup \{(root, u) : \forall u \in U\}$
    $g \leftarrow (V, E, L_V)$
  end if
end for

The generated graph data now is a set of one weakly connected graph for each product development project. Such a set can be directly used as input to many FSM algorithms. However, some FSM algorithms only support one single graph as input. In such a case, it is necessary to modify the graph data and connect all individual graphs in the set. This can be accomplished by creating a new all-encompassing graph G_{all} that contains all individual per-project graphs. Then a single root node is inserted into G_{all} and connected to the root node of each per-project graph.

Third Step: FSM

Now that the input is prepared for common varieties of FSM, it can be used to find all frequent subgraphs that are present in the input graph data. This step of the proposed algorithm will run FSM iteratively until a user-supplied number n of subgraphs was found, as outlined in Algorithm 5.3.

The resulting n subgraphs have the highest support possible in the input graph data. To achieve this, the algorithm uses the minimum support min_supp parameter common to all varieties of FSM. Minimum support defines the number of times a subgraph must occur in order to be a result of FSM. Thus, it restricts the resulting subgraphs to those that are frequent among the development projects. By dynamically adjusting this value in several FSM runs, the algorithm will iteratively arrive at the user-specified number of n subgraphs. For the first run of FSM, the algorithm sets the minimum support to the amount of graphs/projects in the input data. For a subgraph to be returned now, it would need to be present in every single project, which is very unlikely considering the unstructured nature of these projects. In the next iteration, the algorithm decreases minimum support and runs FSM again. Subgraphs returned by FSM are collected in a set of results. This loop is repeated until the set of results contains as many subgraphs as desired.

Algorithm 5.3 Discover process patterns in product development projects. Third step: FSM.

$n \leftarrow Input()$ ▷ Desired number of results, user-supplied
$min\_supp \leftarrow |graphs|$
$subgraphs \leftarrow \emptyset$
repeat
  $results \leftarrow FSM(graphs, min\_supp)$
  $subgraphs \leftarrow subgraphs \cup results$
  $min\_supp \leftarrow min\_supp - 1$
until $|subgraphs| \geq n$

Fourth Step: Post-processing

The n discovered subgraphs represent the desired process patterns in product development projects. For further use, the resulting set of subgraphs may be ranked or otherwise sorted, e.g., by the size of the subgraphs. This post-processing step of the proposed algorithm is outlined in Algorithm 5.4. Especially when the returned result set is large, it may be advisable to focus on those subgraphs with larger size and, therefore, significance. Larger subgraphs, especially when with a high support, have more explanatory power

than smaller ones because they can explain a larger part of the development projects. Depending on the use case, it may be feasible to define an explicit cutoff value, e.g., the minimum size a subgraph needs to have to be useful. In such a case, smaller subgraphs may be dropped from the result set.

Algorithm 5.4 Discover process patterns in product development projects. Fourth step: post-processing.

$minimum\_size \leftarrow Input()$ ▷ Minimum size of subgraphs, user-supplied
$subgraphs \leftarrow sort(subgraphs)$
return $\{s : s \in subgraphs \wedge |s| \geq minimum\_size\}$

Depending on the FSM algorithm used, the discovered subgraphs are purely structural patterns and a discovered pattern is identical in each graph. However, as previously explained in Section 5.3.2, there are also algorithms with other definitions of patterns. These tolerate uncertainty in cases with high variations in unstructured patterns, or noise and incompleteness in cases where data are missing or incorrect.

5.4 Evaluation

This section presents the evaluation of the proposed approach by comparing the results of different FSM algorithms and by analyzing the respectively discovered patterns. Three key points are the primary interest:

1. Can the proposed approach discover existing patterns in the data.
2. How many patterns of what size can it find.
3. How well the FSM algorithms cope with noisy data.

To this end, the evaluation generates randomized input data that mimics unstructured processes and embeds variants of predefined base patterns. Then, the evaluation applies FSM algorithms and evaluates whether it can discover these variants of the base patterns. Generating randomized data also allows degrading the data quality in a controlled way, e.g., to simulate noise or incompleteness. This way, the evaluation may discover the limitations of specific FSM algorithms. The following subsections describe the data generation in

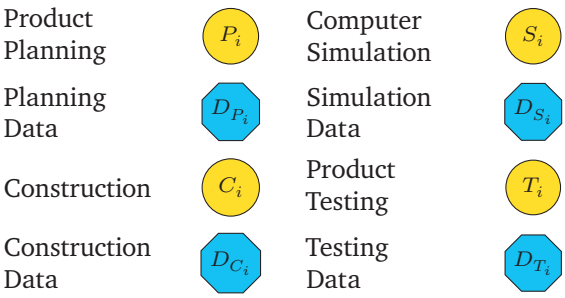

Figure 5.4: Types of activities (yellow) and data (blue) used by the data generation.

Section 5.4.1, followed by the experimental setup in Section 5.4.2, and a discussion of the results in Section 5.4.3.

5.4.1 Data Generation

The evaluation uses a generator for synthetic data sets to create graphs that resemble product development projects. The data generator uses a set of high-level activities that also exist at the industry partner of this thesis, are part of their product development, and are consistent with literature [HDGL13]. It considers the activities and the data they produce as described in Section 5.1 and as summarized in Figure 5.4. These activities are: product planning, computer simulation, prototype construction, and product testing.

Based on these activities and data, the data generation needs to know which combinations of activity and data are valid. Each activity node X_i may have directed edges pointing towards an arbitrary amount of data nodes D_{X_i} that represent the data produced by this activity. An activity node also may have incoming edges from arbitrary data nodes, representing which data this activity requires as input. An edge may not directly connect two activity nodes with no data node in-between.

To form graphs representing product development projects, the data generation uses an algorithmic approach that contains random choices. As starting point, the evaluation supplies two predefined base patterns to the data generation, which are depicted in Figure 5.5. The pattern on the left begins with a

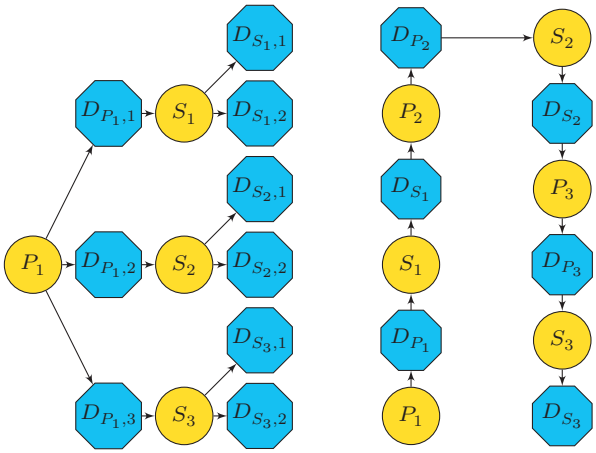

Figure 5.5: Example patterns for parallel (left) and sequential (right) processes in product development [ZRS+22].

product planning activity P_1 that produces three virtual product plans $D_{P_1,i}$. Each of these plans undergoes a computer simulation S_p to assess product characteristics that are then stored in simulation results $D_{S_p,q}$. This pattern emphasizes development projects with multiple activities performed in parallel. The pattern on the right shows a sequential version of a similar project. It emphasizes the iterative improvement of one virtual product based on previous simulation results. So, it contains several sequentially executed planning steps (P_i) and simulation activities (S_p).

Based on these patterns, the data generation grows large graphs with varying sizes and structures, as outlined in Listing 5.1. To grow a graph, the data generation distinguishes between *active* and *inactive* nodes. Initially, all nodes with no outgoing edge are considered active. The data generation algorithm in Listing 5.1 receives a graph as input and stores its nodes with no outgoing edge in a double-ended queue Q. Nodes in this queue are the active nodes. Subsequently, each active node continuously passes through three consecutive decision points: termination, splitting, and growth.

In *termination*, each node makes a binary decision whether to remain active and continue with the other two decisions (see Listing 5.1, line 7). When a node decides not to remain active anymore, it is no longer pro-

cessed, i.e., it no longer passes through these decision points. The probability for termination, i.e., for a node deciding to be inactive, is calculated with $max(0.2, \sqrt{new\_nodes/max\_new\_nodes})$, where new_nodes is the number of newly created nodes for this graph so far and max_new_nodes is a constant limiting the maximum size of a graph, which is set to 2000.

In *splitting*, each active node decides on the number of outgoing edges that are subsequently created for this node (see Listing 5.1, line 11). The data generation algorithm draws the number of edges from a shifted normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$. This makes one outgoing edge the most likely case, with decreasing probabilities for higher numbers of edges, while the maximum number of edges per active node in the generated data is 5.

In *growth*, each node decides which new node type to create for each previously created outgoing edge (see Listing 5.1, line 14). If the currently growing node is a data node, the data generation algorithm creates a new activity node. Here, there is an equal probability for this node to be one of the four types of activities, e.g., product planning, computer simulation, prototype construction, and product testing (see Figure 5.4). If the currently growing node is an activity, the data generation algorithm creates a data node with the same type of the parent activity node, e.g., a computer simulation gets a simulation result as output data node. Newly created nodes are active by default, so that the data generation algorithm for generating new nodes is subsequently applied to them as well. The generation of the whole graph is finished once no active node remains.

Each of these graphs is a unique and randomly generated instance resembling an unstructured process and embedding variants of one of the two base patterns (see Figure 5.5). The data generation algorithm can steer the structure of the graph based on the mentioned probability parameters. Termination limits the size of a graph and depends on the number of generated nodes. With the parameters as previously described, the probability of termination is at least 20% when a new graph is started and contains no nodes yet. This probability then increases rapidly with the creation of new nodes and decreases afterwards, similar to a logistic function. Termination and splitting influence how long paths in the graph can get and how often a graph branches out. The choice of

```
def grow_graph(graph : Node) -> Node:
    new_nodes : int = 0
    max_new_nodes : int = 2000
    Q : deque[Node] = deque(leaves(graph))
    while Q:
        u = Q.popleft()
        # termination: do we terminate this node?
        term = max(0.2, math.sqrt(new_nodes /
         ↪  max_new_nodes))
        if random.random() < term:
            continue
        # splitting: determine number of new nodes
        splits = round(0.5 + abs(random.normalvariate(0,
         ↪  1)))
        new_nodes += splits
        # growth: determine node types for new nodes
        for _ in range(splits):
            node : Node
            if u.node_type is ACTIVITY:
                node = Data_Node(u.action_type)
            if u.node_type is DATA:
                action_type =
                 ↪  random.choice(list(Action_Type))
                node = Activity_Node(action_type)
            u.outgoing.append(node)
            Q.append(node)
    return graph
```

Listing 5.1: Generation of one randomized graph resembling a product development project.

probabilities here favors longer, less broad graphs. This corresponds to general product development projects. Within a development project, product variants are developed in long, uninterrupted sequences with occasional branches for alternatives or revisions. The equally likely choice of each node type during growth ensures that large, unintended patterns do not form randomly.

To further align the synthetic data set with realistic development projects, the data generation introduces errors into the generated graphs in a post-

processing step. The data quality is degraded in two phases: addition of random nodes and deletion of random nodes. When a node is deleted, all its outgoing edges are connected randomly to other nodes with outgoing edges to the deleted node to keep the graph connected. This may result in a graph that does no longer adhere to the previous definitions of valid graphs, e.g., when an activity node is directly connected to another activity. This way, the data generation can simulate actual errors in data, e.g., due to human errors in data documentation. The data generation subsequently increase these errors by changing the frequency of random additions and deletions of nodes. This way, the evaluation may investigate how the patterns found by FSM algorithms change.

5.4.2 Experimental Setup

To mine frequent patterns, the evaluation uses one of the most cited FSM algorithms, gSpan [YH02], and a more recent proposal, GraMi [EASK14]. With default parameters, GraMi searches for frequent patterns similar to gSpan. Additionally, it supports a user-defined distance threshold δ to tune the detection of patterns. For instance, with $\delta = 3$, two connected nodes of a pattern may be matched with two nodes that are up to three hops away from each other. This allows finding patterns with tolerance for small structural differences, e.g., missing edges. However, this drastically increases computational overhead and required hardware resources. Thus, the evaluation combines a distance threshold $\delta = 2$ with GraMi's approximation mode, called AGraMi. AGraMi does not return invalid patterns, but it may miss a few valid frequent patterns. This can be controlled with a user-defined parameter $0 < \alpha \leq 1$. AGraMi with $\alpha = 1$ is equivalent to GraMi. A lower value for α reduces computational overhead, but increases approximation error. This evaluation uses $\alpha = 2 \cdot 10^{-4}$, which is still a fairly high value, results in hardware requirements comparable to GraMi, and performed well for the authors of the algorithm [EASK14].

While gSpan expects its input as a collection of several graphs, GraMi expects one single, large graph. The data generation prepared modified versions of the generated data for each algorithm, in accordance with the second step of the proposed approach illustrated in Section 5.3.3. The evaluation uses an available

Python-based implementation of gSpan[1] and a Java-based implementation of GraMi[2] published by the authors themselves. With both gSpan and GraMi, the evaluation searched for patterns in multiple runs with a varying minimum support parameter. The investigated minimum support values range from 60 to 100 in increments of 10. The maximum is 100, as this corresponds to the number of occurrences of the base patterns used for data generation. With decreasing minimum support, the number of discovered patterns increases. The evaluation stops at 60 as a reasonable minimum that still allows manually verifying the resulting number of patterns. To verify the stability of pattern mining with respect to errors in the input data, the evaluation also performed these experiments on data with 1%, 5%, and 10% of errors. 1% means that each existing node in a graph had a probability of 1% for two different kinds of error: addition of a new node and deletion of itself.

The graph generation and the application of both FSM algorithms is implemented via Python 3 scripts[3]. This also includes a visualization of discovered patterns to verify these patterns visually. The data generation created a data set with in total 200 graphs, i.e., with 100 graphs per base pattern (see Figure 5.5), each with an average of 300 nodes. All steps were executed on a system running Ubuntu 20.04 LTS, an Intel®Xeon®CPU E3-1505M with 4 cores, and 32 GB of system memory.

5.4.3 Results

Figure 5.6 shows the results for this evaluation for generated graphs without errors. Each of these plots visualizes how many patterns of a certain size have been discovered by each algorithm. The size of the discovered patterns is plotted on the x-axis as number of nodes and the y-axis shows how many unique patterns of this size were found. Note that the y-axis has a logarithmic scale. In each plot, the upper-right corner indicates the minimum support, i.e., $s = 100, 90, 80$, and 60.

For $s = 100$, both gSpan and GraMi, discover exactly one pattern with a size of 13. This is the largest discovered pattern and matches exactly the left base

[1] gSpan: https://github.com/betterenvi/gSpan
[2] GraMi: https://github.com/ehab-abdelhamid/GraMi
[3] Github: https://github.com/IPVS-AS/graphgen

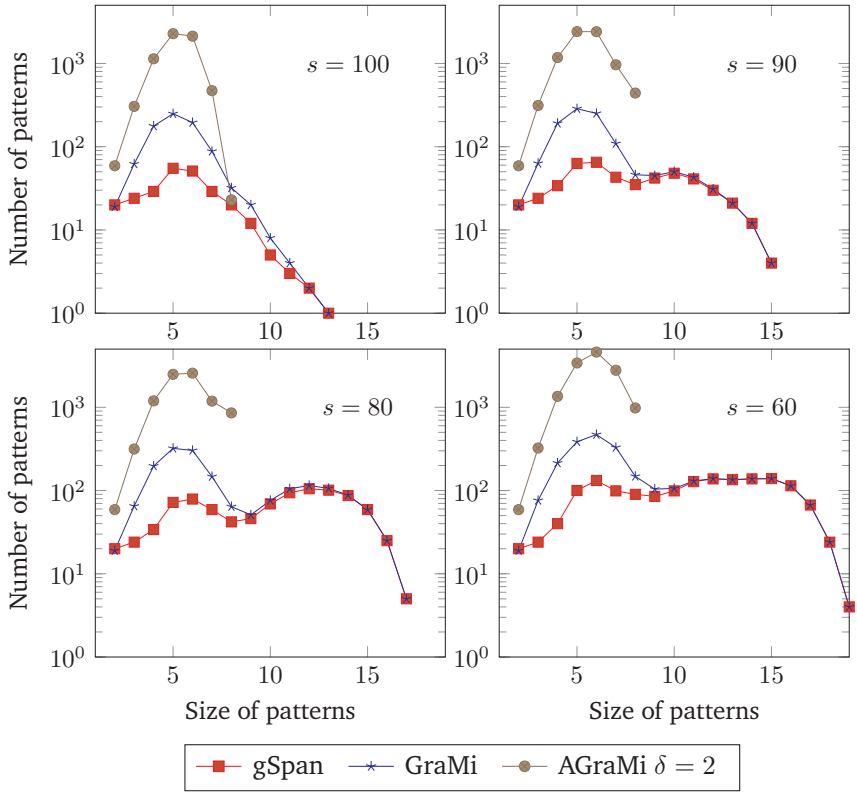

Figure 5.6: Correlation between number and size of discovered patterns. Minimum support s indicated in the upper right [ZRS+22].

pattern in Figure 5.5. Both discover two patterns of size 12. The first of these two patterns is the right pattern in Figure 5.5 and the second one constitutes a subset of the left pattern with one node missing. The algorithms identify this subset of the left pattern as another valid pattern because each subgraph of another subgraph with support s also has the same or higher support and is, thus, a valid result, only smaller. Consequently, the number of discovered patterns increases for smaller patterns because there may be many patterns of size $k < n$ in a pattern of size n.

The number of discovered patterns peaks at a size of 5 and decreases with both, smaller and larger patterns. This is due to different effects that influence

the maximum possible number of unique patterns per size. For patterns with a very small size, the maximum possible number of unique patterns is low. As the size of the patterns increases, i.e., the number of nodes in a pattern increases, the number of possible unique patterns also increases due to the higher number of possible combinations. However, as the size of the patterns increases, the probability that these patterns formed in the randomly generated graph decreases. Due to both effects, the number of patterns found initially increases the larger the patterns become. At a certain point, the number of patterns found starts to decrease, as their size makes it increasingly unlikely that they were generated. The position of the peak is influenced by the number of node types as this directly influences the number of possible combinations for patterns. To verify this, multiple additional graph datasets were generated and subsequently analyzed with all three FSM algorithms. Instead of four types of activity and data nodes, as used in the dataset evaluated here (see Figure 5.4), graphs with 5, 10, and 15 types of activity and data nodes were also investigated. These tests confirmed that the position of the peak of discovered patterns correlates with the number of node types in the graph data.

For smaller pattern sizes, GraMi discovers more patterns than gSpan. The evaluation visually confirmed that these additional patterns were indeed valid results. The higher number is due to duplicates in the results of GraMi, while gSpan deduplicates the results better. AGraMi discovered substantially more patterns due to its relaxed pattern definition with distance threshold $\delta = 2$, which also matches nodes up to two hops away. As with GraMi and gSpan, the reason for AGraMi's immense peak is the number of unique combinations of nodes, only that its distance threshold further amplifies the effect. For pattern sizes greater than 5 nodes, AGraMi converges to values similar to GraMi and gSpan. However, AGraMi did not discover patterns larger than 8 nodes. This is due to the approximation parameter $\alpha = 2 \cdot 10^{-4}$, which lets AGraMi miss valid patterns of greater size, as a trade-off for improved performance. A higher value may discover larger patterns, but this was not feasible on the used hardware due to memory constraints.

For the lower support values $s = 90$, $s = 80$, and $s = 60$, the evaluation observes an increase in the number of larger patterns. Visual inspection confirms that these patterns of size larger than 13 nodes all include the base

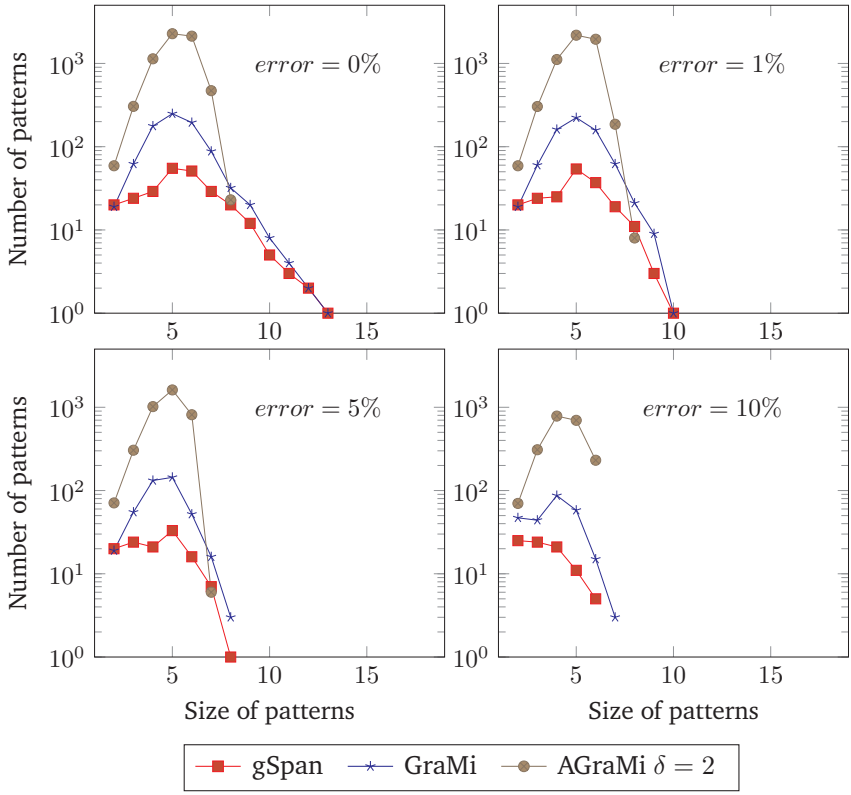

Figure 5.7: Correlation between number and size of discovered patterns with a minimum support of 100, but with varying error rates as indicated in the upper right [ZRS+22].

patterns (see Figure 5.5). The data generation created graphs by algorithmically growing these base patterns into larger graphs. With four general types of activity nodes, it is likely for specific extended variants of the base patterns to be generated in multiple graphs simply by chance. As the value for the minimum support decreases, more of these larger subgraphs are identified by the algorithms as valid frequent patterns. Note that GraMi and gSpan produce identical results for patterns larger than 12 in all cases with $s < 100$, and, thus, their plots partially overlap in Figure 5.6.

Figure 5.7 also shows the correlation between the number and size of discovered patterns. Here, the evaluation kept minimum support constant at 100 and instead continuously degraded the data quality with two different kinds of error: adding erroneous nodes and deleting existing nodes. The introduced error is shown in the upper-right corner and corresponds to both kinds of errors separately. For instance, $error = 1\%$ means that each node had a 1% probability to gain a new erroneous outgoing node and additionally had a 1% probability to get deleted. So, for each node up to two errors can happen, both with a separate probability of 1%. Note that the degraded data are based on the same set of graphs used previously, not a newly generated random data set.

For comparison purposes, the upper-left diagrams in Figure 5.6 and Figure 5.7 are identical, i.e., with a minimum support $s = 100$ and $error = 0\%$. It can be seen that for $error = 1\%$ (upper-right diagram in Figure 5.7), the largest pattern found drops to a size of 10 for both gSpan and GraMi. This discovered pattern is a subset of the left base pattern (see Figure 5.5), i.e., three nodes of this base pattern are missing in the pattern identified by the algorithms. Visually, the identified pattern had a very close resemblance to the base pattern and can, thus, be recognized without any problems. With increasing $error$, the largest patterns found become smaller and smaller. Nevertheless, even for a high $error = 10\%$, the discovered patterns with a length of up to 7 nodes still visually resemble the base patterns. AGraMi could not discover larger patterns than gSpan and GraMi despite its distance threshold parameter because it was limited by its approximation parameter. For $error = 5\%$ and $error = 10\%$, the largest detected pattern of AGraMi (size 7 and 6, respectively) was smaller than or equal to the results of gSpan (8 and 6) and GraMi (8 and 7). Nevertheless, AGraMi again found many more patterns with smaller sizes.

5.4.4 Discussion

Based on these results, this section summarizes the most important differences of the results obtained by the three algorithms for frequent subgraph mining:

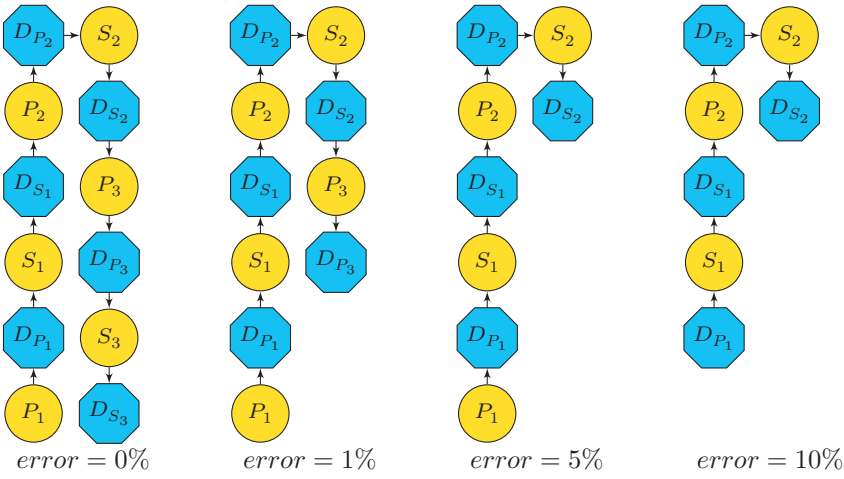

Figure 5.8: The four largest sequential patterns that were discovered in data sets with 0%, 1%, 5%, and 10% errors, respectively, from left to right. The higher the error, the shorter the patterns [ZRS+22].

- For data without error, GraMi and gSpan perform similarly and succeed both in discovering the base patterns. GraMi discovers more variants of these patterns of smaller size than gSpan.
- AGraMi identifies even more patterns of smaller size, but it struggles to identify longer patterns. This is due to AGraMi's approximation parameter that limits the length of patterns that can be discovered. Hence, this parameter needs to be tuned carefully for a given use case and its development projects.
- For data with errors, GraMi and gSpan are not able to identify longer patterns anymore, i.e., the length of the longest discovered patterns decreases proportionally to the error rate.
- AGraMi can better handle noisy data due to its distance threshold. However, this requires significant computing resources.

The following is a summary of the major findings with respect to the three key points stated in the beginning of Section 5.4.

First, the evaluation can successfully discover the base patterns or subsets of it. Patterns with high minimum support should be preferred to ensure the significance of patterns. For this reason, the proposed approach to discover patterns in product development projects (see Section 5.3.3 Third Step) starts with a high value for minimum support and iteratively decreases it when not enough patterns have been found. Regarding the second key point, the number of discovered patterns increases as the size of discovered patterns decreases. Therefore, discovered patterns may be ranked by size and pruned when they are smaller than a specific size (see Section 5.3.3 Fourth Step). Third, degraded data negatively impacts the ability of the tested algorithms to discover large patterns. FSM algorithms that are more tolerant to noise may be desirable for future investigations. Nevertheless, the algorithms still found subgraphs that visually resemble the base patterns shown in Figure 5.5. For instance, Figure 5.8 shows the four largest sequential patterns with support $s = 100$ that were discovered in the data sets with 0%, 1%, 5%, and 10% $errors$, respectively. In case of no error, the FSM algorithms discovered exactly the sequential base pattern. With increasing error, the length of the longest discovered pattern decreases. However, even with a very high $error = 10\%$, the result visually resembles the sequential base pattern.

All these identified patterns shown in Figure 5.8 exhibit a noticeable lack of parallelism. In the example in Figure 5.1, the intention is to parallelize work activities to reduce time to completion in development projects. The discovery of many sequential patterns shows that this parallel project execution was not fully achieved in practice. An organization can now use this discovery as a cue to investigate why development projects deviate from the intended parallelism and instead show many sequential sub-processes. Subsequently, the organization can start to optimize the execution of its projects.

5.5 Summary

This chapter proposes a novel approach to process discovery for product development projects. Product development uses information systems, such as PLM systems, to store data and track the lifecycle of products. These systems contain rich information on provenance and dataflow dependencies between

activities in product development projects. This rich information gets lost when transforming the data to the event log format used in process discovery. To overcome this, this chapter instead proposes a graph-based data structure that is able to represent activities and data of product development as well as dataflow dependencies. Based on this data structure, frequent subgraph mining discovers patterns of sub-processes in product development projects. Such patterns can better express different variants and common subsets of unstructured processes. That is another major advantage over classic process discovery, which often wrongly assumes the existence of a single, generic process model.

A prototypical implementation evaluates this novel approach with a prototype based on generated data that resemble typical product development projects. This allows objectively verifying the discovered patterns and shows that existing FSM algorithms are indeed able to identify common patterns reflecting typical sub-processes in unstructured product development projects. Such discovered patterns represent process knowledge and may guide inexperienced engineers in their decision-making.

Chapter 6

Conclusion and Future Work

Companies with strong activities in the field of virtual product development generate large amounts of heterogeneous data [HDGL13]. The heterogeneity in the structure and format of virtual product development data makes it difficult for domain experts to discover and analyze them [PBC+13]. Yet, control over data discovery and analysis tasks should be in the hands of domain experts, as stated by Viaene [Via13].

In this context, this thesis has formulated several research challenges (see Section 1.2) that prevent domain experts from harnessing the full value of their data. Section 6.1 provides a summary of the research contributions of this thesis (see Chapter 3, 4, and 5). Section 6.2 presents future work that builds on the research contributions of this thesis.

6.1 Summary of the Contributions

This thesis focuses on virtual product development data and the harnessing of these data for cross-organizational data analysis by domain experts through metadata. To this end, research challenges RC_1, RC_2, and RC_3 were identified in Section 1.2 of this thesis, which were then addressed with research contributions. These research contributions are summarized below.

C_1 A Metadata Model to Connect Data in Virtual Product Development [ZRKM20; ZRKM21]

This contribution addresses research challenge RC_1 in Chapter 3 and proposes a metadata model that offers a tailored approach to managing data and metadata in virtual product development.

The main aspect of this contribution is the metadata model Graph'MeMo. This model is the first to connect the heterogeneous data formats of virtual product development across data silos in a way that is highly accessible to domain experts. Graph'MeMo represents data of virtual product development in a graph structure that also represents the work activities of product development projects. Work activities are all activities in these projects that produce or consume data and are the actual work performed either by an employee or a machine through computer software. As the central element of a metadata model, Graph'MeMo can attribute and describe both data and work activities with metadata in the same way. Thus, the metadata model provides the means to describe all data and work activities of product development projects in one interconnected structure.

Furthermore, Graph'MeMo represents the relationships between data and work activities. This explicit representation of data and their relationships to work activities and metadata natively represents the dataflow of development projects as it is familiar to domain experts. Domain experts can use these relationships for data discovery, data exploration, and subsequent data analyses. Additionally, they can also use this information to more easily compare and reproduce their data and experiments.

For application to a real-world use case and to validate its feasibility, a reference architecture was designed that combines the metadata model with a data lake. The concept of a data lake is well suited for the prevailing heterogeneity of data formats in virtual product development. Because of the abstraction of the data by the metadata model, it is possible to represent data sets together in the same interconnected structure, even if they are physically stored in different storage backends.

The application and implementation of a case study shows that the metadata model with implementation in the reference architecture was capable of realizing the industrial use case. The metadata model was shown to be able to

meet the challenges of data in virtual product development. The implementation successfully connects different organizational units and their data silos with heterogeneous data formats. The work activities as a first-class entity with its own metadata as well as the representation of the data flow allowed domain experts to independently discover and select highly specific data sets for data analyses. The case study showed that data selection benefited from the metadata of the data set itself, but also from the metadata of the work activities that generated and processed the data sets.

C_2 Integration of Domain Experts into Data Analyses [ZRK+24]

This contribution addresses research challenge RC_2 in Chapter 4 and describes a generic approach for the creation of data-driven surrogate models with domain experts integrated into data analysis projects.

The main aspect of this contribution is to integrate domain experts and their knowledge into a process that is primarily driven by data scientists: data analysis. Feature engineering has a notable impact on the quality of a predictive model. At the same time, feature engineering in virtual product development requires background knowledge of physical processes that a data scientist often does not have. Therefore, this contribution focuses on how to integrate domain experts and their knowledge into the process of data analysis.

To this end, this contribution analyzed a real-world use case involving the creation of surrogate models for long-running computer simulations. Based on this use case, two types of domain knowledge were identified that are suitable to be integrated into the machine learning pipeline: scientific knowledge and expert knowledge. Scientific Knowledge describes fundamental knowledge about physical principles and other engineering disciplines. This knowledge can be expressed in mathematical form. Expert Knowledge is knowledge that comes from the experience and intuition of the domain expert. This is less formal and, therefore, must be contributed through human feedback.

The proposed approach consists of a machine learning pipeline with six steps that integrates both types of knowledge from domain experts. The approach combines domain experts interactively participating in these steps, formal knowledge of domain experts expressed by algebraic expressions, and existing methods from the field of Auto ML. A case study demonstrates the application

of this approach and evaluates its results. For this purpose, the approach was implemented and applied to a real-world data set. Graph'MeMo is used for data selection because selecting suitable training data sets has a high impact on prediction quality. With Graph'MeMo, domain experts can apply their domain knowledge, and discover as well as select the most appropriate data sets for training a particular surrogate model. In order to make a meaningful comparison, a baseline was also implemented, which only omits the feature engineering part with integrated domain experts, but still performs all other optimizations. The case study shows that the prediction models trained by the presented approach have better prediction quality as well as higher reliability. Furthermore, the physical correctness of the surrogate models is ensured, while still benefiting from an automatable, data-driven machine learning pipeline.

C_3 Process Discovery in Virtual Product Development [ZRS+22]

This contribution addresses research challenge RC_3 in Chapter 5 and proposes a novel approach to process discovery.

The main aspect of this contribution is the representation of product development projects as graphs and the use of Frequent Subgraph Mining (FSM) to search for patterns in the graphs of these projects. These patterns can be used by companies to identify their real-world processes. This is necessary to identify issues with the company processes and optimize them. However, classic algorithms of process discovery are not well suited for virtual product development because there are no event logs and the processes are very heterogeneous due to human decision-making.

Therefore, this contribution approaches the problem of process discovery in a different way. Instead of using heuristics to approximate a process model based on event logs, this contribution uses Graph'MeMo and the provenance of the product development projects already contained in the metadata structure. Graph'MeMo represents data and work activities as well as the dataflow between them as performed by engineers. With product development projects represented as graphs, FSM algorithms can be used to search for patterns in these projects.

The proposed approach consists of five steps and, in the end, yields a set of patterns with a configurable significance. Based on a representation of product

development projects as individual graphs, as can be done using Graph'MeMo, pre-processing takes place first. The individual project graphs are connected in such a way that they belong to a common, large graph. This graph represents the input for the FSM algorithm. Before this algorithm can be executed, the user determines a desired support value. This defines in how many projects the searched pattern must be present to be considered significant. In a final post-processing step after the execution of the FSM algorithm, all patterns found that are too short and, therefore, of little use are discarded.

An implementation was created to evaluate this approach. In addition to the described approach, a data generator was also created to generate graphs that mimic the unstructured processes of product development. This strategy allows the targeted embedding of predefined patterns. By means of these, it can be objectively shown how well the FSM algorithms can find these patterns in the data. In order to also consider the reality of noisy data sets and human input errors, different types of errors were also introduced into the data. This allows an assessment of how well the FSM algorithms can handle these problems. The results attest that the tested FSM algorithms can find the embedded patterns in the data. When it comes to data with errors, the examined algorithms perform differently, which highlights the importance of choosing the right FSM algorithm.

6.2 Future Work

Despite the challenges and contributions addressed in this thesis, much more work remains. With the goal of cross-departmental data analytics in virtual product development, other important issues include further empowering domain experts through data management and tools, and democratizing data access, e.g., through self-service data marketplaces. Specifically with regard to the contributions presented here, the following open points remain, among others:

Automated Graph Generation

To maximize the added value of a metadata structure such as Graph'MeMo, it should be used consistently in all product development processes involved.

However, it is a hindrance if the graph structure has to be created manually. In addition, human data entry also leaves room for error. Accordingly, the graph structure should be built automatically. Computer software used by domain experts should generate relevant nodes and associate them with each other. However, this requires customization of the often proprietary software in virtual product development. In addition, metadata should be automatically extracted from existing files and data sets. In the context of this thesis, this issue has been briefly explored. Finding a solution that works generically for a set of heterogeneous data formats is a non-trivial and complex task.

Improve Access for Domain Experts

While the graph structure is explicitly designed for intuitive exploration by domain experts, further tooling can add value here. In the context of this thesis, suitable visualization concepts were tested, such as the direct visualization of the metadata structure as a graph, as well as a processed visualization that corresponded to a data catalog. It became clear early on that the type of visualization that is more suitable depends heavily on the use cases and the level of knowledge of the domain expert. These open points remain and need further exploration. That is, which is the optimal visualization for which use case, or which level of training offers the optimal balance for organizations.

Another aspect concerns the further integration of domain experts in data analysis projects. Here, too, additional tools may help to better support the domain expert in data exploration and add domain knowledge during data analysis.

Recommendation System for Product Development Projects

With patterns discovered in product development projects, various follow-up activities may be performed. First, the patterns can be compared with guidelines predefined by an organization for the execution of development projects. The discrepancy between reality and theory presents potential for improvement and optimization in the execution of development projects. Additionally, the discovered patterns can be examined to determine whether they correlate with successful or unsuccessful projects. With the patterns appropriately labeled, a

recommendation system for development projects can then be trained. This can assist the developers in the course of a project and either support their decision-making, or provide hints when negative patterns are detected.

List of Author Publications

[1] J. Ziegler, P. Reimann, F. Keller, and B. Mitschang. "A Graph-based Approach to Manage CAE Data in a Data Lake." In: *Procedia CIRP: 53rd CIRP Conference on Manufacturing Systems (CIRP CMS)*. Volume 93. Elsevier, 2020, pages 496–501. DOI: 10.1016/j.procir.2020.04.155.

[2] J. Ziegler, P. Reimann, F. Keller, and B. Mitschang. "A Metadata Model to Connect Isolated Data Silos and Activities of the CAE Domain." In: *Proceedings of the 33rd International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer International Publishing, 2021, pages 213–228. ISBN: 978-3-030-79382-1. DOI: 10.1007/978-3-030-79382-1_13.

[3] J. Ziegler, P. Reimann, C. Schulz, F. Keller, and B. Mitschang. "A Graph Structure to Discover Patterns in Unstructured Processes of Product Development." In: *Proceedings of the 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE, 2022, pages 222–227. DOI: 10.1109/IRI54793.2022.00057.

[4] T. Gose, C. Schulz, F. Keller, J. Ziegler, and C. Hitzke. "Smart Filter Element Development — Applied Machine Learning for Optimized Element Layouts." In: *Proceedings of FILTECH*. 2023. ISBN: 978-3-941655-21-8.

[5] J. Ziegler, P. Reimann, A. Kilian, C. Schulz, F. Keller, and B. Mitschang. "An Approach to Integrate Domain Knowledge into Feature Engineering to Enhance Data-Driven Surrogate Models of Simulations." In: *Procedia CIRP: 57th CIRP Conference on Manufacturing Systems (CIRP CMS)*. Volume 130. Elsevier, 2024, pages 1276–1281.

Supervised Student Theses and Projects

[1] F. Niebling. “Approximation aufwendiger Simulationen durch datengetriebene Surrogatmodelle.” Bachelor Thesis. Universität Stuttgart, Feb. 2020.

[2] P. Bredl, J. Greif, M. Erdemann, and S. Weiler. “Visualisierung und Exploration eines Datenbestandes mit graphbasierter Metadatenverwaltung.” Research Project. Universität Stuttgart, June 2020.

[3] T. Boceck. “A Software Architecture for Graph-based Metadata Management for CAE Data.” Master Thesis. Universität Stuttgart, Nov. 2020.

[4] D. Moslavac. “Konzeption einer Ontologie zur Beschreibung von Daten der CAE Domäne.” Bachelor Thesis. Universität Stuttgart, Feb. 2021.

[5] R. Münch. “Konzeption und Realisierung eines Werkzeugs zur Extraktion von Metadaten aus einem heterogenen Datenbestand des Computer-Aided Engineering.” Bachelor Thesis. Universität Stuttgart, June 2021.

[6] A. Steinhauer. “Konzeption und Realisierung eines konfigurierbaren Werkzeugs zur Visualisierung von Metadaten.” Bachelor Thesis. Universität Stuttgart, June 2021.

Bibliography

[ABD+14] M. Allanic, T. Brial, A. Durupt, M. Joliot, P. Boutinaud, and B. Eynard. "Towards an Enhancement of Relationships Browsing in Mature PLM Systems." In: *Product Lifecycle Management for a Global Market*. Berlin, Heidelberg: Springer, 2014, pages 345–354. ISBN: 978-3-662-45937-9 (cited on pages 24, 102, 110, 112, 117).

[AG08] R. Angles and C. Gutierrez. "Survey of Graph Database Models." In: *ACM Compute Survey* 40.1 (Feb. 2008). ISSN: 0360-0300. DOI: 10.1145/1322432.1322433 (cited on pages 49, 68).

[AvvW04] A. Alves De Medeiros, B. van Dongen, W. M. van der Aalst, and A. Weijters. *Process Mining: Extending the Alpha-Algorithm to Mine Short Loops*. BETA Working Paper Series. Technische Universiteit Eindhoven, 2004. ISBN: 90-386-1978-2 (cited on page 108).

[AW16] Y. Anang and Y. Watanabe. "Applying Layering Concept to the Software Requirements Analysis and Architectural Design." In: *REFSQ Workshops*. 2016 (cited on pages 53–55).

[Awa07] A. Awad. "BPMN-Q: A Language to Query Business Processes." In: *Enterprise Modelling and Information Systems Architectures - Concepts and Applications, Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*. Edited by M. Reichert, S. Strecker, and K. Turowski. Volume P-119. LNI. GI, 2007, pages 115–128. URL: https://dl.gi.de/20.500.12116/22195 (cited on page 45).

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 2003 (cited on page 53).

[Boc20] T. Boceck. “A Software Architecture for Graph-based Metadata Management for CAE Data.” Master Thesis. Universität Stuttgart, Nov. 2020 (cited on page 58).

[Bos02] Bose, R. “A Conceptual Framework for Composing and Managing Scientific Data Lineage.” In: *Proc. of the 14th International Conference on Scientific and Statistical Database Management*. Edinburgh, Scotland, UK, July 2002, pages 15–19. DOI: 10.1109/SSDM.2002.1029701 (cited on page 41).

[BWD+21] A. Birk, Y. Wilhelm, S. Dreher, C. Flack, P. Reimann, and C. Gröger. “A Real-World Application of Process Mining for Data-Driven Analysis of Multi-Level Interlinked Manufacturing Processes.” In: *Procedia CIRP* 104 (2021). 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0, pages 417–422. ISSN: 2212-8271. DOI: 10.1016/j.procir.2021.11.070. URL: https://www.sciencedirect.com/science/article/pii/S2212827121009689 (cited on page 21).

[BWK+22] A. Blum, Y. Wilhelm, S. Klein, C. Schnur, P. Reimann, R. Müller, and A. Schütze. “Ganzheitlicher Ablaufplan für wissensgetriebene Projekte des maschinellen Lernens in der Produktion.” In: *tm - Technisches Messen* 89.5 (2022), pages 363–383. DOI: 10.1515/teme-2022-0027 (cited on page 80).

[CCF+09] C. E. Catalano, E. Camossi, R. Ferrandes, V. Cheutet, and N. Sevilmis. “A Product Design Ontology for Enhancing Shape Processing in Design Workflows.” In: *Journal of Intelligent Manufacturing* 20.5 (2009), pages 553–567. DOI: 10.1007/s10845-008-0151-z (cited on page 49).

[CKC20] Y. Cui, S. Kara, and K. C. Chan. “Manufacturing Big Data Ecosystem: A Systematic Literature Review.” In: *Robotics and Computer-Integrated Manufacturing* 62 (2020), page 101861. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2019.101861. URL: https://www.sciencedirect.com/science/article/pii/S0736584519300559 (cited on pages 19, 20).

[Cou+12] N. R. Council et al. *Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification*. National Academies Press, 2012. ISBN: 978-0309256346 (cited on pages 17, 38).

[CSP13] D. Che, M. Safran, and Z. Peng. “From Big Data to Big Data Mining: Challenges, Issues, and Opportunities.” In: *Proc. of the 18th International Conference on Database Systems for Advanced Applications*. Edited by B. Hong, X. Meng, L. Chen, W. Winiwarter, and W. Song. Wuhan, China: Springer Berlin Heidelberg, 2013, pages 1–15. ISBN: 978-3-642-40270-8 (cited on pages 20, 38, 40).

[CYZH07] C. Chen, X. Yan, F. Zhu, and J. Han. “gApprox: Mining Frequent Approximate Patterns from a Massive Network.” In: *7th IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pages 445–450. DOI: 10.1109/ICDM.2007.36 (cited on page 116).

[DF08] S. B. Davidson and J. Freire. “Provenance and Scientific Workflows: Challenges and Opportunities.” In: *Proc. of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD)*. Vancouver, Canada, 2008, pages 1345–1350. ISBN: 9781605581026. DOI: 10.1145/1376616.1376772 (cited on pages 19, 41, 110).

[Dix10] J. Dixon. *Pentaho, Hadoop, and Data Lakes*. 2010. URL: https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/ (cited on page 30).

[Dom12] P. Domingos. “A Few Useful Things to Know about Machine Learning.” In: *Commun. ACM* 55.10 (Oct. 2012), pages 78–87. ISSN: 0001-0782. DOI: 10.1145/2347736.2347755 (cited on page 81).

[DRG20] S. Dreher, P. Reimann, and C. Gröger. “Application Fields and Research Gaps of Process Mining in Manufacturing Companies.” English. In: *Proceedings of INFORMATIK 2020*. Edited by R. H. Reussner, A. Koziolek, and R. Heinrich. GI Gesellschaft für Informatik e.V. (GI), Oct. 2020, pages 621–634. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2020-57&engl=1 (cited on page 101).

[dSMF+19] C. dos Santos Garcia, A. Meincheim, E. R. Faria Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, and E. E. Scalabrin. “Process Mining Techniques and Applications - A Systematic Mapping Study.” In: *Expert Systems with Applications* 133 (2019), pages 260–295. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2019.05.003 (cited on pages 21, 101, 106).

[EASK14] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. “GraMi: Frequent Subgraph and Pattern Mining in a Single Large Graph.” In: *Proc. VLDB Endow.* 7.7 (2014), pages 517–528. ISSN: 2150-8097. DOI: 10.14778/2732286.2732289 (cited on pages 110, 115, 116, 126).

[Fan15] H. Fang. “Managing Data Lakes in Big Data Era: What’s a Data Lake and Why Has It Became Popular in Data Management Ecosystem.” In: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. June 2015, pages 820–824. DOI: 10.1109/CYBER.2015.7288049 (cited on pages 31, 46, 61).

[GE03] I. Guyon and A. Elisseeff. “An Introduction to Variable and Feature Selection.” In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pages 1157–1182. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944968 (cited on page 81).

[GGH+19] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, and B. Mitschang. “Leveraging the Data Lake: Current State and Challenges.” In: *Big Data Analytics and Knowledge Discovery*. Edited by C. Ordonez, I.-Y. Song, G. Anderst-Kotsis, A. M. Tjoa, and I. Khalil. Cham: Springer International Publishing, 2019, pages 179–188. ISBN: 978-3-030-27520-4. DOI: 10.1007/978-3-030-27520-4_13 (cited on page 31).

[GHP10] R. Gabriel, T. Hoppe, and A. Pastwa. “Classification of Metadata Categories in Data Warehousing - A Generic Approach.” In: *AMCIS 2010 Proceedings*. 2010. URL: https://aisel.aisnet.org/amcis2010/133 (cited on page 32).

[GLN+05] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber. “Scientific Data Management in the Coming Decade.” In: *SIGMOD Record* 34.4 (Dec. 2005), pages 34–41. ISSN: 0163-5808. DOI: 10.1145/1107499.1107503 (cited on pages 19, 20, 40, 41).

[GT08] M. W. Grieves and M. Tanniru. “PLM, Process, Practice and Provenance: Knowledge Provenance in Support of Business Practices in Product Lifecycle Management.” In: *International Journal of Product Lifecycle Management* 3.1 (2008), pages 37–53. DOI: 10.1504/IJPLM.2008.019969 (cited on pages 21, 104–106, 111, 118).

[Gv07] C. W. Günther and W. M. van der Aalst. “Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics.” In: *Business Process Management*. Berlin, Heidelberg: Springer, 2007, pages 328–

343. ISBN: 978-3-540-75183-0 (cited on pages 22, 106, 107, 109, 114).

[Hao+21] J. Hao et al. “Building Surrogate Models for Engineering Problems by Integrating Limited Simulation Data and Monotonic Engineering Knowledge.” In: *Advanced Engineering Informatics* 49 (2021). DOI: 10.1016/j.aei.2021.101342. URL: https://www.sciencedirect.com/science/article/pii/S1474034621000951 (cited on page 82).

[HDB17] M. Herschel, R. Diestelkämper, and H. Ben Lahmar. “A Survey on Provenance: What for? What Form? What From?” In: *VLDB Journal* 26 (Dec. 2017), pages 881–906. ISSN: 1066-8888. DOI: 10.1007/s00778-017-0486-1 (cited on pages 19, 41, 111).

[HDGL13] M. Hirz, W. Dietrich, A. Gfrerrer, and J. Lang. “Overview of Virtual Product Development.” In: *Integrated Computer-Aided Design in Automotive Development: Development Processes, Geometric Fundamentals, Methods of CAD, Knowledge-Based Engineering Data Management*. Berlin, Heidelberg: Springer, 2013, pages 25–50. ISBN: 978-3-642-11940-8. DOI: 10.1077/978-3-642-11940-8_2 (cited on pages 17, 18, 21, 36, 104, 111, 122, 135).

[HFM06] A. Halevy, M. Franklin, and D. Maier. “Principles of Dataspace Systems.” In: *Proceedings of the Twenty-Fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’06. Chicago, IL, USA: Association for Computing Machinery, 2006, pages 1–9. ISBN: 1595933182. DOI: 10.1145/1142351.1142352 (cited on page 30).

[HGQ16] R. Hai, S. Geisler, and C. Quix. “Constance: An Intelligent Data Lake System.” In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD ’16. San Francisco, California, USA: Association for Computing Machinery, 2016, pages 2097–2100. ISBN: 9781450335317. DOI: 10.1145/2882903.2899389 (cited on page 32).

[HKV19] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Cham: Springer International Publishing, 2019, page 219. ISBN: 978-3-030-05317-8. DOI: 10.1007/978-3-030-05318-5 (cited on pages 87, 88, 94).

[HMPR04] A. Hevner, S. March, J. Park, and S. Ram. *Design Science in Information Systems Research*. 2004 (cited on page 22).

[HMvdW+20] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." In: *Nature* 585.7825 (Sept. 2020), pages 357–362. DOI: 10.1038/s41586-020-2649-2 (cited on page 89).

[Hol16] A. Holzinger. "Interactive machine learning for health informatics: when do we need the human-in-the-loop?" In: *Brain Informatics* 3.2 (June 2016), pages 119–131. ISSN: 2198-4026. DOI: 10.1007/s40708-016-0042-6 (cited on page 75).

[HPST12] G. Herzwurm, W. Pietsch, S. Schockert, and T. Tauterat. "QFD for Cloud Computing." In: *Proceedings of the 18th International Symposium on Quality Function Deployment. International Symposium on Quality Function Deployment (ISQFD)*. 2012 (cited on page 53).

[HRT+23] V. Hirsch, P. Reimann, D. Treder-Tschechlov, H. Schwarz, and B. Mitschang. "Exploiting Domain Knowledge to address Class Imbalance and a Heterogeneous Feature Space in Multi-Class Classification." English. In: *International Journal on Very Large Data Bases (VLDB-Journal)* (2023). DOI: 10.1007/s00778-023-00780-6 (cited on page 80).

[HSG+17] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, K. Ramachandran, S. Arora, A. Bhattacharyya, S. Das, M. Donsky, G. Fierro, C. She, C. Steinbach, V. Subramanian, and E. Sun. "Ground: A Data Context Service." In: *Proc. of the 8th Biennial Conf. on Innovative Data Systems Research*. Chaminade, CA, USA, 2017 (cited on pages 20, 42, 44, 71).

[HSM97] G. Herzwurm, S. Schockert, and W. Mellis. *Qualitätssoftware durch Kundenorientierung: die Methode Quality Function Deployment (QFD); Grundlagen, Praxis und SAP R/3 Fallbeispiel*. Springer, 1997 (cited on page 52).

[HSRW12] M. Hoppen, M. Schluse, J. Roßmann, and B. Weitzig. "Database-driven Distributed 3D Simulation." In: *Proceedings of the 2012 Winter Simulation Conference (WSC)*. Dec. 2012, pages 1–12. DOI: 10.1109/WSC.2012.6465091 (cited on pages 20, 44).

[HST15] G. Herzwurm, S. Schockert, and T. Tauterat. “Quality Function Deployment in Software Development -State-of-the-art.” In: *Proceedings of the 21th International Symposium on Quality Function Deployment*. 2015 (cited on pages 52, 53).

[HZC21] X. He, K. Zhao, and X. Chu. “AutoML: A survey of the state-of-the-art.” In: *Knowledge-Based Systems* 212 (2021), page 106622. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2020.106622. URL: https://www.sciencedirect.com/science/article/pii/S0950705120307516 (cited on pages 19, 21, 74, 88).

[JCZ13] C. Jiang, F. Coenen, and M. Zito. “A Survey of Frequent Subgraph Mining Algorithms.” In: *The Knowledge Engineering Review* 28.1 (2013), pages 75–105. DOI: 10.1017/S0269888912000331 (cited on pages 114–116).

[JQ17] M. Jarke and C. Quix. “On Warehouses, Lakes, and Spaces: The Changing Role of Conceptual Modeling for Data Integration.” In: *Conceptual Modeling Perspectives*. Edited by J. Cabot, C. Gómez, O. Pastor, M. R. Sancho, and E. Teniente. Cham: Springer International Publishing, 2017, pages 231–245. ISBN: 978-3-319-67271-7. DOI: 10.1007/978-3-319-67271-7_16 (cited on pages 30, 46).

[JZS20] P. Jiang, Q. Zhou, and X. Shao. *Surrogate Model-Based Engineering Design and Optimization*. Singapore: Springer, 2020. ISBN: 978-981-15-0730-4. DOI: 10.1007/978-981-15-0731-1 (cited on page 73).

[Kra02] K. Krawiec. “Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks.” In: *Genetic Programming and Evolvable Machines* 3 (2002), pages 329–343. ISSN: 1573-7632. DOI: 10.1023/A:1020984725014 (cited on page 81).

[LFv13] S. J. J. Leemans, D. Fahland, and W. M. van der Aalst. “Discovering Block-Structured Process Models from Event Logs - A Constructive Approach.” In: *Application and Theory of Petri Nets and Concurrency*. Berlin, Heidelberg: Springer, 2013, pages 311–329. ISBN: 978-3-642-38697-8 (cited on pages 108, 109).

[LFv16] S. J. J. Leemans, D. Fahland, and W. M. van der Aalst. “Using Life Cycle Information in Process Discovery.” In: *Business Process Management Workshops*. Cham, Switzerland: Springer International Publishing, 2016, pages 204–217. ISBN: 978-3-319-42887-1 (cited on page 107).

[LPS18] R. Lukyanenko, J. Parsons, and V. C. Storey. “Modeling Matters: Can Conceptual Modeling Support Machine Learning?” In: *AIS SIGSAND* (2018), pages 1–12 (cited on pages 80, 86).

[Mar83] J. Martin. *Managing the Data-Base Environment*. International Conference on ENTERprise Information Systems (CENTERIS). Prentice Hall PTR, 1983. ISBN: 0-13-550582-8 (cited on page 56).

[Mat17] C. Mathis. “Data Lakes.” In: *Datenbank-Spektrum* 17.3 (Nov. 2017), pages 289–293. ISSN: 1610-1995. DOI: 10.1007/s13222-017-0272-7 (cited on pages 30, 31, 46).

[MBvdA12] F. M. Maggi, R. P. J. C. Bose, and W. M. P. van der Aalst. “Efficient Discovery of Understandable Declarative Process Models from Event Logs.” In: *Advanced Information Systems Engineering*. Edited by J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pages 270–285. ISBN: 978-3-642-31095-9 (cited on page 109).

[McK10] W. McKinney. “Data Structures for Statistical Computing in Python.” In: *Proceedings of the 9th Python in Science Conference*. Edited by S. van der Walt and J. Millman. 2010, pages 56–61. DOI: 10.25080/Majora-92bf1922-00a (cited on page 89).

[Mil13] J. J. Miller. “Graph Database Applications and Concepts with Neo4j.” In: *Proceedings of the Southern Association for Information Systems Conference*. Atlanta, GA, USA, 2013 (cited on page 62).

[ML16] C. Madera and A. Laurent. “The next Information Architecture Evolution: The Data Lake Wave.” In: *Proceedings of the 8th International Conference on Management of Digital EcoSystems*. MEDES. Biarritz, France: Association for Computing Machinery, 2016, pages 174–180. ISBN: 9781450342674. DOI: 10.1145/3012071.3012077 (cited on pages 30, 31).

[MSF+10] B. Matthews, S. Sufi, D. Flannery, L. Lerusse, T. Griffin, M. Gleaves, and K. K. van Dam. “Using a Core Scientific Metadata Model in Large-Scale Facilities.” In: *Int. Journal of Digital Curation* 5.1 (2010), pages 106–118. DOI: 10.2218/ijdc.v5i1.146 (cited on pages 20, 44, 71).

[MT16] N. Miloslavskaya and A. Tolstoy. “Big Data, Fast Data and Data Lake Concepts.” In: *Procedia Computer Science* 88 (2016). 7th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2016, held July 16 to July 19, 2016 in New York City, NY, USA, pages 300–305. ISSN: 1877-0509. DOI: 10.1016/j.procs.2016.07.439. URL: https://www.sciencedirect.com/science/article/pii/S1877050916316957 (cited on page 31).

[Nie20] F. Niebling. “Approximation aufwendiger Simulationen durch datengetriebene Surrogatmodelle.” Bachelor Thesis. Universität Stuttgart, Feb. 2020 (cited on page 89).

[NK05] S. Nijssen and J. N. Kok. “The Gaston Tool for Frequent Subgraph Mining.” In: *Electronic Notes in Theoretical Computer Science* 127.1 (2005), pages 77–87. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2004.12.039 (cited on page 116).

[NMMA16] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen. *Microservice Architecture: Aligning Principles, Practices, and Culture*. ”O’Reilly Media, Inc.”, 2016 (cited on page 59).

[Nus01] B. Nuseibeh. “Weaving the Software Development Process Between Requirements and Architecture.” In: *From Software Requirements to Architectures* (2001) (cited on page 54).

[Obj14] Object Management Group. *Business Process Model and Notation*. 2014. URL: https://www.omg.org/spec/BPMN/ (cited on pages 44, 45, 71).

[PBC+13] L. C. Pouchard, M. L. Branstetter, R. B. Cook, R. Devarakonda, J. Green, G. Palanisamy, P. Alexander, and N. F. Noy. “A Linked Science investigation: enhancing climate change data discovery with semantic technologies.” In: *Earth Science Informatics* 6.3 (Sept. 2013), pages 175–185. ISSN: 1865-0481. DOI: 10.1007/s12145-013-0118-2 (cited on page 135).

[PCR19] A. M. Pirbazari, A. Chakravorty, and C. Rong. “Evaluating Feature Selection Methods for Short-Term Load Forecasting.” In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. Feb. 2019, pages 1–8. DOI: 10.1109/BIGCOMP.2019.8679188 (cited on pages 81, 90).

[PIS11] O. Papapetrou, E. Ioannou, and D. Skoutas. “Efficient Discovery of Frequent Subgraph Patterns in Uncertain Graph Databases.” In: *Proceedings of the 14th International Conference on Extending Database Technology*. Sweden: ACM, 2011, pages 355–366. ISBN: 9781450305280. DOI: 10.1145/1951365.1951408 (cited on page 116).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. “Scikit-learn: Machine Learning in Python.” In: *Journal of Machine Learning Research* 12 (2011), pages 2825–2830 (cited on pages 89–91).

[QHV16] C. Quix, R. Hai, and I. Vatov. “Metadata Extraction and Management in Data Lakes With GEMMS.” In: *Complex Systems Informatics and Modeling* 9 (2016), pages 67–83. DOI: 10.7250/csimq.2016-9.04 (cited on pages 20, 43, 44, 71).

[Ric15] M. Richards. *Microservices vs. Service-oriented Architecture*. O’Reilly Media Sebastopol, 2015 (cited on page 58).

[Ril17] J. Riley. “Understanding Metadata: What is Metadata, and What is it For?” In: *National Information Standards Organization* (Jan. 2017). URL: http://www.niso.org/publications/press/UnderstandingMetadata.pdf (cited on page 32).

[RN10] M. A. Rodriguez and P. Neubauer. “Constructions from Dots and Lines.” In: *Bulletin of the American Society for Information Science and Technology* 36.6 (2010), pages 35–41. DOI: 10.1002/bult.2010.1720360610 (cited on page 50).

[RRS+11] P. Reimann, M. Reiter, H. Schwarz, D. Karastoyanova, and F. Leymann. “SIMPL - a framework for accessing external data in simulation workflows.” In: *Datenbanksysteme für Business, Technologie und Web (BTW)*. Bonn: Gesellschaft für Informatik e.V., 2011, pages 534–553. ISBN: 978-3-88579-274-1 (cited on page 47).

[RSM14] P. Reimann, H. Schwarz, and B. Mitschang. “A Pattern Approach to Conquer the Data Complexity in Simulation Workflow Design.” In: *Proc. of the 22nd International Conference on Cooperative Information Systems (CoopIS)*. Edited by R. M. et al. Amantea, Italy: Springer-Verlag, 2014, pages 21–38 (cited on page 47).

[RZ19] F. Ravat and Y. Zhao. “Metadata Management for Data Lakes.” In: *New Trends in Databases and Information Systems*. Edited by T. Welzer, J. Eder, V. Podgorelec, R. Wrembel, M. Ivanović, J. Gamper, M. Morzy, T. Tzouramanis, J. Darmont, and A. Kamišalić Latifić. Cham: Springer International Publishing, 2019, pages 37–44. ISBN: 978-3-030-30278-8 (cited on page 32).

[SD21] P. Sawadogo and J. Darmont. “On data lake architectures and metadata management.” In: *Journal of Intelligent Information Systems* 56.1 (Feb. 2021), pages 97–120. ISSN: 1573-7675. DOI: 10.1007/s10844-020-00608-7 (cited on page 32).

[Som10] I. Sommerville. *Software Engineering*. 9th. USA: Addison-Wesley Publishing Company, 2010. ISBN: 0137035152 (cited on page 53).

[SP16] I. Suriarachchi and B. Plale. “Crossing analytics systems: A case for integrated provenance in data lakes.” In: *2016 IEEE 12th International Conference on e-Science (e-Science)*. 2016, pages 349–354. DOI: 10.1109/eScience.2016.7870919 (cited on pages 19, 41).

[SSF+19] P. Sawadogo, É. Scholly, C. Favre, É. Ferey, S. Loudcher, and J. Darmont. “Metadata Systems for Data Lakes: Models and Features.” In: *Proc. of the Workshop on BBIGAP at the 23rd European Conference on Advances in Databases and Information Systems*. Springer, 2019. DOI: 10.1007/978-3-030-30278-8_43 (cited on pages 20, 42, 44, 71).

[TCW98] C.-L. Tsai, Z. Cai, and X. Wu. “The examination of residual plots.” In: *Statistica Sinica* 8.2 (Jan. 1998), pages 445–465. ISSN: 1017-0405. URL: http://www3.stat.sinica.edu.tw/statistica/j8n2/j8n29/j8n29.htm (cited on pages 81, 96).

[TWP+19] E. Trunzer, I. Weiß, T. Pötter, C. Vermum, M. Odenweller, S. Unland, D. Schütz, and B. Vogel-Heuser. “Big Data trifft Produktion: Neun Pfeiler der industriellen Smart-Data-Analyse.” In: *atp magazin* 61.1-2 (2019), pages 90–98 (cited on pages 21, 30, 46, 74, 80).

[van+12] W. M. van der Aalst et al. “Process Mining Manifesto.” In: *Business Process Management Workshops*. Berlin, Heidelberg: Springer, 2012, pages 169–194. ISBN: 978-3-642-28108-2 (cited on pages 21, 101).

[van11a] W. M. van der Aalst. “On the Representational Bias in Process Mining.” In: *2011 IEEE 20th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. 2011, pages 2–7. DOI: 10.1109/WETICE.2011.64 (cited on pages 107, 108).

[van11b] W. M. van der Aalst. “Process Mining: Discovering and Improving Spaghetti and Lasagna Processes.” In: *2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. 2011. DOI: 10.1109/CIDM.2011.6129461 (cited on pages 105, 107, 110, 114).

[van11c] W. M. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Germany: Springer, 2011. ISBN: 978-3-642-19344-6 (cited on pages 106–109).

[van12] W. M. van der Aalst. “Process Mining: Overview and Opportunities.” In: *Transactions on Management Information Systems* 3.2 (2012). ISSN: 2158-656X. DOI: 10.1145/2229156.2229157 (cited on page 106).

[van19] W. M. van der Aalst. “A Practitioner’s Guide to Process Mining: Limitations of the Directly-Follows Graph.” In: *Procedia Computer Science* 164 (2019), pages 321–328. ISSN: 1877-0509. DOI: 10.1016/j.procs.2019.12.189 (cited on page 113).

[VEC04] C. Vasilakis, E. El-Darzi, and P. Chountas. “A Data Warehouse Environment for Storing and Analyzing Simulation Output Data.” In: *Proceedings of the 2004 Winter Simulation Conference, 2004.* Volume 1. Dec. 2004, page 710. DOI: 10.1109/WSC.2004.1371379 (cited on pages 20, 44, 71).

[VFP+05] M. Valle, J. Favre, E. Parkinson, A. Perrig, and M. Fahrat. “Scientific Data Management for Visualization Implementation Experience.” In: *Simulation und Visualisierung 2005 (SimVis 2005)*. Magdeburg, Germany: SCS Publishing House e.V., 2005, pages 347–354 (cited on pages 20, 44, 71).

[Via13] S. Viaene. “Data Scientists Aren’t Domain Experts.” In: *IT Professional* 15.6 (2013), pages 12–17. ISSN: 1941-045X. DOI: 10.1109/MITP.2013.93 (cited on pages 19, 21, 135).

[VK11] S. Vinodh and D. Kuttalingam. “Computer-aided Design and Engineering as Enablers of Agile Manufacturing.” In: *Journal of Manufacturing Technology Management* 22 (3 2011). ISSN: 1741-038X. DOI: 10.1108/17410381111112747 (cited on pages 17, 18).

[vRMB+23] L. von Rueden, S. Mayer, K. Beckh, B. Georgiev, S. Giesselbach, R. Heese, B. Kirsch, J. Pfrommer, A. Pick, R. Ramamurthy, M. Walczak, J. Garcke, C. Bauckhage, and J. Schuecker. "Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems." In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2023), pages 614–633. DOI: 10.1109/TKDE.2021.3079836 (cited on page 83).

[vWM04] W. M. van der Aalst, T. Weijters, and L. Maruster. "Workflow Mining: Discovering Process Models from Event Logs." In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (2004), pages 1128–1142. DOI: 10.1109/TKDE.2004.47 (cited on pages 108, 113).

[WGW12] C. Willis, J. Greenberg, and H. White. "Analysis and Synthesis of Metadata Goals for Scientific Data." In: *Journal of the American Society for Information Science and Technology* 63.8 (2012), pages 1505–1520. DOI: 10.1002/asi.22683 (cited on page 20).

[WSR+20] Y. Wilhelm, U. Schreier, P. Reimann, B. Mitschang, and H. Ziekow. "Data Science Approaches to Quality Control in Manufacturing: A Review of Problems, Challenges and Architecture." In: *Service-Oriented Computing*. Edited by S. Dustdar. Cham: Springer International Publishing, 2020, pages 45–65. ISBN: 978-3-030-64846-6 (cited on pages 19, 21, 74).

[WvWS07] L. Wen, W. M. van der Aalst, J. Wang, and J. Sun. "Mining Process Models with non-Free-Choice Constructs." In: *Data Mining and Knowledge Discovery* 15.2 (2007), pages 145–180 (cited on page 108).

[Yan+21] L. Yang et al. "B-PINNs: Bayesian Physics-Informed Neural Networks for Forward and Inverse PDE Problems with Noisy Data." In: *Journal of Computational Physics* 425 (2021). DOI: 10.1016/j.jcp.2020.109913. URL: https://www.sciencedirect.com/science/article/pii/S0021999120306872 (cited on page 82).

[YH02] X. Yan and J. Han. "gSpan: Graph-based Substructure Pattern Mining." In: *IEEE International Conference on Data Mining*. 2002, pages 721–724. DOI: 10.1109/ICDM.2002.1184038 (cited on pages 110, 115, 116, 126).

[ZLGZ09] Z. Zou, J. Li, H. Gao, and S. Zhang. “Frequent Subgraph Pattern Mining on Uncertain Graph Data.” In: *Proceedings of the 18th Conference on Information and Knowledge Management*. CIKM. Hong Kong, China: ACM, 2009, pages 583–592. ISBN: 9781605585123. DOI: 10.1145/1645953.1646028 (cited on page 116).

[ZRK+24] J. Ziegler, P. Reimann, A. Kilian, C. Schulz, F. Keller, and B. Mitschang. “An Approach to Integrate Domain Knowledge into Feature Engineering to Enhance Data-Driven Surrogate Models of Simulations.” In: *Procedia CIRP: 57th CIRP Conference on Manufacturing Systems (CIRP CMS)*. Volume 130. Elsevier, 2024, pages 1276–1281 (cited on pages 75, 137).

[ZRKM20] J. Ziegler, P. Reimann, F. Keller, and B. Mitschang. “A Graph-based Approach to Manage CAE Data in a Data Lake.” In: *Procedia CIRP: 53rd CIRP Conference on Manufacturing Systems (CIRP CMS)*. Volume 93. Elsevier, 2020, pages 496–501. DOI: 10.1016/j.procir.2020.04.155 (cited on pages 22, 24, 36, 74, 80, 82, 92, 136).

[ZRKM21] J. Ziegler, P. Reimann, F. Keller, and B. Mitschang. “A Metadata Model to Connect Isolated Data Silos and Activities of the CAE Domain.” In: *Proceedings of the 33rd International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer International Publishing, 2021, pages 213–228. ISBN: 978-3-030-79382-1. DOI: 10.1007/978-3-030-79382-1_13 (cited on pages 22, 24, 36, 37, 44, 48, 51, 57, 63–65, 71, 74, 82, 92, 105, 110, 112, 136).

[ZRS+22] J. Ziegler, P. Reimann, C. Schulz, F. Keller, and B. Mitschang. “A Graph Structure to Discover Patterns in Unstructured Processes of Product Development.” In: *Proceedings of the 23rd International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE, 2022, pages 222–227. DOI: 10.1109/IRI54793.2022.00057 (cited on pages 19, 22, 102–104, 112, 116, 123, 128, 130, 132, 138).

[Zul95] R. E. Zultner. “Software QFD: The North American Experience.” In: *Proceedings of the 1st Pacific Rim Symposium on Quality Deployment*. 1995, pages 163–174 (cited on page 53).

All URLs were last checked on 01. 07. 2023.

List of Figures

List of Tables

List of Listings

List of Algorithms