Multidimensional process discovery

Beta

Research School for Operations Management and Logistics

Multidimensional Process Discovery

Joel Ribeiro

Multidimensional Process Discovery

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Ribeiro, Joel T.S.

Multidimensional Process Discovery / by Joel T.S. Ribeiro.
- Eindhoven: Technische Universiteit Eindhoven, 2013. - Proefschrift. -

A catalogue record is available from the Eindhoven University of Technology Library

ISBN: 978-90-386-3341-1
NUR: 982
Keywords: Process Mining / Process Discovery / Enhancement / OLAP / Data Mining

The research described in this thesis has been carried out under the auspices of Beta Research School for Operations Management and Logistics.

This work has been carried out as part of the project "Merging of Incoherent Field Feedback Data into Prioritized Design Information (DataFusion)", sponsored by the Dutch Ministry of Economic Affairs, Agriculture and Innovation under the IOP IPCR program.

Beta Dissertation Series D165

Printed by Proefschriftmaken.nl || Uitgeverij BOXPress
Cover design: Proefschriftmaken.nl || Uitgeverij BOXPress

Multidimensional Process Discovery

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op donderdag 13 maart 2013 om 16.00 uur

door

Joel Tiago Soares Ribeiro

geboren te Vila de Cucujães, Portugal

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. P.W.P.J. Grefen

Copromotor:
dr. A.J.M.M. Weijters

To my family

Contents

Chapter 1

Introduction

Business process management (BPM) is increasingly becoming present in modern enterprises. This fact is explained by the orientation shift within organizations, from data orientation to process orientation. Process aware information systems have been used to manage the execution of business processes, providing support especially at the operational level. Business process intelligence (BPI) techniques such as process mining can be applied to get strategic insight into the business processes. Process *discovery*, *conformance* checking and *enhancement* are possible applications for knowledge discovery on process data.

Typically represented in event logs, business process data describe the execution of process events over time. This means that operational data are associated with process events, which turns the static nature of the business data into dynamic. This is a great advantage for business process analysis once that the business behavior can be tracked. By applying process mining techniques on data that describe the behavior of process instances, it is possible to discover and analyze the business as it is being executed. However, so far, these techniques are typically designed to focus on specific process dimensions, omitting information potentially relevant for the analysis comprehension.

Specially designed to support on-the-fly hypothesis-driven exploration of data, online analytic processing (OLAP) systems are commonly used as reporting tools in almost every application for business intelligence. Exploiting the data by combining different dimensions with some measures of interest, it is possible to adjust on-the-fly the analysis' level of abstraction in an interactive way. Relying on the multidimensional data model, OLAP tools organize the data in such a way that it is possible to have multiple perspectives in the same analysis.

This thesis presents a research work concerning the extension of process mining techniques with OLAP functionalities. Making use of OLAP concepts, a multidimensional data structure is designed to hold the different dimensions of business processes in such a way that process discovery and analysis are facilitated. Extending the traditional OLAP data cubes, these structures can improve the process analysis by providing immediate results under different levels of abstraction. Furthermore, non-trivial process patterns and relations can also be identified by exploiting the different process perspectives.

The research work presented in this thesis was conducted under the scope of the DataFusion project, which is described next.

1.1 DataFusion Project

The DataFusion project aims at the extraction of structural, richer and prioritized field feedback information from the existing New Product Development (NPD) operational databases. Fusing data from these databases can be potentially used to enrich the information in the new NPD data warehouses. Together with the input from product developers, information related to mismatches between product specifications and customer requirements can be obtained.

> *The goal of the DataFusion project is to support product development with prioritized information related to mismatches between product specifications and customer requirements.*

The DataFusion project, in long term, aims at preventing consumer complaints through product development. Thus, in order to achieve this objective, it is necessary to develop new consumer information processes. These processes should be organized in such way that the relevant information can be easily identified and, consequently, prioritized design information can be generated. The hypothesis is that this information can be generated by combining the consumer information from the existing NPD datasets. Assuming the validity of this hypothesis, this project concerns the design of a new consumer information repository based on existing NPD databases in order to deal with the mismatched information. This system should be able to establish an effective connection between the data sources and the knowledge discovery techniques. Three different aspects of this system are studied in the following subprojects.

Information Integration and Distribution in NPD focuses on generating the information requirements and developing a new information process to provide development with the required information.

Information Extraction from Unstructured Texts focuses on the development of techniques for efficiently and accurately extracting relevant information from large amount of texts.

Knowledge Discovery on NPD Information focuses on the development of suitable knowledge discovery techniques that can be applied to identify various patterns on NPD information.

The focus of this thesis is the knowledge discovery on NPD datasets (the third subproject described above). Therefore, the contribution of this thesis – to the DataFusion project – consists of (i) the design of a concept system to establish an effective connection between the data sources and the knowledge discovery techniques, and (ii) the development of a framework for knowledge discovery on NPD information.

1.1.1 New Product Development

New Product Development (NPD) can be defined as the complete process of introducing a new product (good or service) to market, from conceptualization to commercialization. The different stages in NPD can be summarized as follows.

Idea Generation and Screening: Spotting good ideas, evaluating them for technical feasibility, marketability and financial viability. It is necessary to involve different departments such as R&D, Production and Sales, and to use information about costumers, competitors, market and others.

Concept Development and Testing: Stating a detailed version of the new product in meaningful consumer terms. Additionally, the new products are tested with a group of consumers to find out whether the concepts have strong consumer appeal.

Business Analysis: Estimating costs, sales, income, break-even point, profit and return on investment (ROI) for the new ideas. It is necessary to find out whether the estimations satisfy the enterprise's goals.

Beta and Market Testing: Turning the idea into a prototype to ensure that the new product is feasible. Additionally, the prototype is tested in typical usage situations. An initial run of the product is produced and sold in a test market area in order to determine customer acceptance.

Technical Implementation: Planning and implementing the product production. Several engineering and production issues need to be solved. Furthermore, there are concerns with design, materials, production processes, quality and safety.

Commercialization: Introducing the new product into the market. Issues such as promotional expenditures at the product launch, penetration prices and distribution plan need to be taken. The timing is critical for success.

Technical and Customer Support: Handling with products malfunctions and customers complaints. This stage does not take part of NPD but its information may be extremely useful and valuable for improving current products or developing better new ones.

Figure 1.1 provides an overview of the NPD process. This process starts with the idea generation for a product and ends with the product commercialization. Technical and customer support is considered as a stepping stone for enabling the development of better products. Field feedback information such as the one that is generated by customer services can be used to characterize the market needs and expectations for new products. Thus, better products can be developed faster.

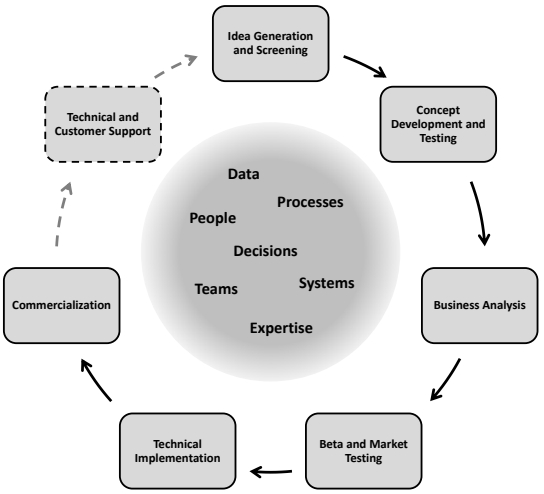


Figure 1.1: Stages in New Product Development.

1.1.2 Field Feedback Information

Field feedback data sources can be rich knowledge repositories for NPD. Nowadays, especially due the Internet, it is easy to share an opinion, experience or complaint about an acquired - or experienced - product or service. When accessible, this information may have a strong influence not only on the consumer audience - which can be potential buyers - but also on the producers. Internet forums, social networks, and blogs are good examples that may work as an immediate kind of word of mouth, affecting market trends and companies goodwill. Other forms of communication, such as the usage of customer services for complaint or request notification, may enable the producers to get good insight into their products in the field and market conditions. This strategic knowledge is an example of what can be extracted from field feedback data sources. However, this task is not trivial but may be achieved through knowledge discovery techniques.

Although knowledge discovery techniques can easily fit in any business context, there are some issues to apply it on field feedback data. The curse of heterogeneity is the major challenge related to information management over the product life cycle. This life cycle generates information under diverse forms (e.g., Internet forums or technical reports), feeding multiple, heterogeneous, and autonomous databases. Moreover, cross-functional teams[1] have multiple, heterogeneous information needs. Thus, the *boundaries*[2] among NPD team members may be seen as an extra constraint for information or knowledge retrieval, especially over heterogeneous databases. Data reliability is a critical issue as well. Unstructured, incoherent and noisy data may lead to misadjusted results. Thus, data integration as well as data enrichment are fundamental tasks to be applied before performing knowledge discovery.

1.2 Knowledge Discovery

Knowledge discovery can be defined as the process of searching large amounts of data in order to find non-trivial patterns (i.e., knowledge). The knowledge discovery domain consists of a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data, in order to support better business decision-making through historical, current, and predictive views of business operations. Knowledge discovery subdomains such as *data mining*, *online analytic processing* (OLAP), and *process mining* have been exhaustively studied over the last decade, offering nowadays a vast amount of techniques that can be used in almost every business context [50, 74, 87].

Although knowledge discovery techniques are easily applicable in any business context according to the literature, there are some issues related to customer experience data that make the application more difficult and complex. The heterogeneous and autonomous character of the customer experience databases is the major challenge for knowledge discovery. The reliability of the data is a critical issue as well. Unstructured, incoherent, and noisy data may lead to misadjusted results. It implies that unstructured customer experience data need to be transformed to structured information before knowledge discovery can be performed.

[1] The NPD process is traditionally handled by cross-functional teams [107].

[2] Concept introduced in [26] to describe the implicit limitation in communication among NPD team members due the absence of similar background skills.

1.2.1 Data Mining

Data mining can be defined as *the extraction of implicit, previously unknown, and potentially useful information from data* [134]. This can be achieved by analyzing automatically large amounts of data in order to find relationships or patterns. Databases, data warehouses, and other repositories are potential data sources for data mining. However, in order to optimize the accuracy of the mining functions, some data preparation such as data integration, transformation and cleaning may be needed before mining. Applications of data mining on field feedback data sources can be found in [61, 71, 84].

Depending on the data mining functionality, different kinds of patterns can be discovered to characterize general aspects of data or to support prediction. The main data mining functionalities can be categorized as follows.

- **Characterization and discrimination** consists of summarizing and comparing general characteristics of data.
- **Rule learning** refers to the discovery of frequent patterns, associations and correlations from data.
- **Classification and estimation** consists of learning models or functions for prediction.
- **Clustering** refers to the process of grouping data objects into classes of similar objects.

Characterization and discrimination is the simplest functionality since it basically relies on statistical functions. One common application of this data mining functionality is described in Subsection 1.2.2. Rule learning, classification and estimation, and clustering are described next.

Rule Learning

Association rule learning [8] consists of finding strong relationships (in form of association rules) in the data. Basically, these relationships are based on frequent patterns (i.e., patterns that occur frequently in data), which can include itemsets, subsequences, and substructures.

- Frequent **itemsets** consist of a set of objects (items) that frequently appear together in data (e.g., a transactional dataset or an event log).
- Frequent **subsequences** consist of a sequence of objects (items) that appear frequently in data.
- Frequent **substructures** consist of structural objects such as subtrees or subgraphs that occur frequently.

The Apriori algorithm [7] is the reference algorithm for mining frequent itemsets. Exploring the Apriori property (*all non-empty subsets of a frequent itemset must also be frequent*), this algorithm finds frequent itemsets by generating candidates from related known frequent itemsets. As an alternative to Apriori, the FP-growth (frequent pattern growth) [51] finds frequent itemsets by using a tree-based structure (an FP-tree) to describe the data. Instead of generating candidates, the FP-growth uses each case in the data to grow the FP-tree and, thus, discover the frequent patterns. Another common approach for mining frequent itemsets is the ECLAT [142], which applies some concepts from information retrieval to efficiently compute the support of the itemsets.

Mining frequent subsequences and substructures can be performed by applying a similar approach as for frequent itemsets. Association rules can be generated directly from frequent patterns. This process is described in detail in Subsection 6.2.3. Eventually, association rules can be extended to correlation rules by using a correlation measure. For an extensive overview of association rule learning techniques see [50, 55, 143].

Classification and Estimation

Classification is a supervised learning technique that can be used to construct a model (or to find a function) that characterizes and discriminates specific categorical aspects (target classes) in data. These classification models are based on a set of training data (i.e., a set of cases with known classes) and can be used to predict the classes in cases when these are not known. A classification model can be represented in various forms depending on the classification approach. These approaches can be characterized as follows.

- **Decision tree induction** is one of the most common classifier techniques. Basically, these techniques consists of constructing a tree-based structure (i.e., a decision tree) where target values (classes) are described according to a training dataset. ID3 [89], C4.5 [90], and CART [24] can be considered as the reference algorithms for the induction of decision trees. Further details about decision tree induction are provided in Chapter 6.
- **Bayesian classification** consists of statistical classifiers based on Bayes' theorem. Naïve Bayesian classifiers [37] and Bayesian belief networks [54] are two common Bayesian classification techniques.
- Artificial **neural networks** are structures mimicking biological neural networks. These structures can be used to model complex relationships between attributes and classes. The backpropagation algorithm [97] can be considered as the reference algorithm for training artificial neural networks.
- A **support vector machine** (SVM) is a classification approach for both linear and non-linear data. Basically, these techniques transform the data in such a way that the multidimensional space can be divided by one or more hyperplanes. The maximum margin training algorithm [21] can be considered as the reference algorithm for computing support vector machines.
- Other classification approaches such as lazy learners [33], genetic algorithms [38], fuzzy set theory [141], among others.

Estimation is a supervised learning technique that can be used to find a function that describes specific continuous values in data. These estimation functions are also based on a set of training data and can be used to predict missing or unknown values. Linear and non-linear regression models [63] are common estimation techniques. Besides regression, several classification techniques can be adapted to predict continuous values (e.g., neural networks and SVM).

The accuracy of a classification model or an estimation function can be assessed by some validation techniques (e.g., holdout, cross-validation, and bootstrap). Basically, the evaluation of a model consists of using a set of testing data (i.e., a set of cases with known classes not used in the learning process) for predicting the known classes. The accuracy assessment can be done by checking the correct and incorrect predictions.

For an extensive overview of classification and estimation techniques see [37, 50, 69].

Clustering

Clustering is an unsupervised learning technique that can be used to group objects into clusters of objects. A cluster can be defined as a collection of objects that are similar to each other but different (dissimilar) from objects in other groups. The measure that define the similarity (or dissimilarity) between objects can be used for learning the clustering models as well as for assessing their quality. Euclidean distance, Minkowski distance, and Jaccard coefficient are examples of these measures. Pattern recognition and outlier detection and analysis are common applications of clustering techniques.

Clustering techniques can be classified into the following categories.

- **Partitioning methods** distribute the objects by a user-specified number of clusters (K). Clusters are typically represented by a centroid that provides the summary description of all objects in the cluster. The K-means algorithm [81] and the K-medoids algorithm [64] are two good examples of partitioning methods.
- **Hierarchical methods** generate a hierarchy of clusters in which the objects are distributed from a single cluster comprising all objects to several one-object clusters. There are two types of hierarchical methods: *agglomerative* and *divisive*. The agglomerative approach starts by considering every object as a cluster and then successively merges the two most similar clusters until only one cluster remains. The divisive approach starts by considering all objects as a single cluster and then successively splits a cluster until all clusters consist of a single object. Examples of both agglomerative and divisive approaches can be found in [64].
- **Density-based methods** are based on the notion of density instead of similarity (or dissimilarity) between objects. The GDBSCAN [98] and the OPTICS [13] algorithms are examples of density-based methods.
- **Grid-based methods** transform the object space into a grid structure with a finite number of cells. The clustering process is then performed over the grid. The STING algorithm [127] is a good example of a grid-based method.
- **Model-based methods** hypothesize different models to define the different clusters and find the best fit of the data to those models. The EM (Expectation-Maximization) [34] and the COBWEB [41] algorithms are examples of model-based methods.

For an extensive overview of clustering techniques see [50, 138].

1.2.2 Online Analytic Processing

Traditionally associated to decision support systems (DSS), OLAP systems provide a different view of the data by applying the multidimensional paradigm [29]. OLAP techniques organize the data under the multidimensional data model. This model represents data by means of a data cube, a multidimensional fact-based structure that supports complex queries in real time. A fact describes an occurrence of a business operation (e.g., sale or purchase), which can be quantified by one or more measures of interest (e.g., the total amount of the sale or purchase) and characterized by multiple dimensions of analysis (e.g., time, location, product and customer). Typically numerical, measures can be aggregated at different levels of abstraction. Nonetheless, the

application of OLAP on non-numerical data is increasingly being explored. Temporal series, graphs, and complex event sequences are possible applications [30, 72, 76]. An application of OLAP in customer service support can be found in [57].

A data cube provides a multidimensional view of data through the computation of specific measures for every combination of the cube's dimension values. Each combination defines a different group of facts and is represented by a *multidimensional cell*. The set of cells that share the same dimensions forms a *cuboid*, which represents a perspective. The complete set of cuboids forms the data cube through a *lattice of cuboids*.

A lot of research have been done to develop efficient methods for computing data cubes. The data cube can be either fully materialized or partially materialized. Full materialization consists of the computation of all cuboid cells in the data cube. Partial materialization consists of the computation of a subset of cuboid cells. Common types of partially materialized data cubes are [50]:

- **Iceberg cubes** [39] simply contain multidimensional cells that satisfy a specific minimum support threshold.
- **Shell cubes** [73] simply contain *shell fragments*. Shell fragments can be defined as a subset of cuboids that are characterized by a specific set of dimensions.
- **Closed cubes** [136] simply contain closed cells. Closed cells can be defined as multidimensional cells that can be used to represent other cells without losing information.

Several efficient materialization methods are proposed. MultiWay [144] is a top-down approach that explores the simultaneous aggregation of multiple dimensions to compute full data cubes. BUC [20] is a bottom-up approach that explores sorting and partitioning for computing iceberg cubes. Comparable to BUC, H-Cubing [52] is a bottom-up approach that uses an efficient data structure (a H-Tree) to compute iceberg cubes. Integrating top-down and bottom-up approaches, Star-Cubing [135, 137] computes either full data cubes or iceberg cubes by performing aggregations on multiple dimensions simultaneously. Shell Cube [73] computes shell fragments based on a selection of dimensions of interest. C-Cubing [136] computes closed cubes by using an algebraic measure that can be computed efficiently and incrementally.

OLAP Mining

Combining OLAP with data mining, OLAP mining (or OLAPing) aims at the discovery of non-trivial patterns from multidimensional views of data. In [48, 49], it is described how the traditional data mining functions can be applied in multidimensional datasets. The cube-gradient analysis problem is introduced in [58]. This problem consists of searching the multidimensional space in order to find *significant changes in measures*. Efficient methods for constrained gradient analysis are proposed in [10, 11, 36].

1.2.3 Process Mining

Process mining can be defined as the *extraction of valuable, process-related information from event logs* [110]. Process aware information systems have been used to manage the execution of business processes, providing support specially at an operational level. Business process intelligence techniques such as process mining can be applied to get

strategic insight into the business processes. Process *discovery*, *conformance* checking and *enhancement* are common applications for knowledge discovery on process data. Figure 1.2 shows an overview about process mining.

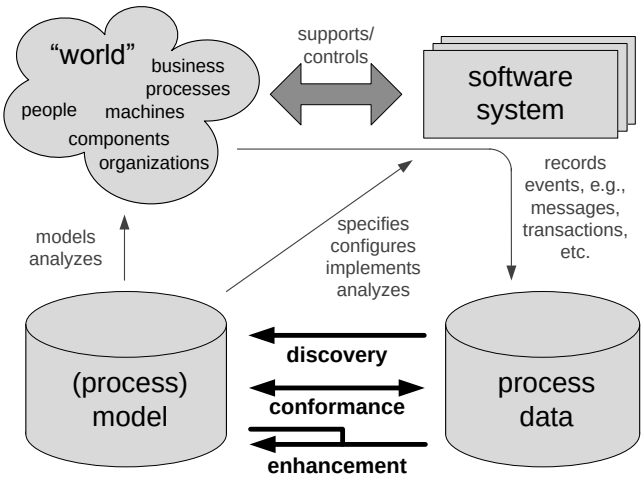


Figure 1.2: Positioning of the main applications of process mining: *discovery*, *conformance*, and *enhancement* [110].

An example of process mining applicability on event-based customer experience data sets can be found in [67].

Process Discovery

Discovery consists of characterizing the behavior of business processes by simply analyzing data about executed processes. This means that, given some process data, the observed behavior described in the data is represented by means of process models. Depending on the discovery technique, process models can be modeled according to several process modeling formalisms. Petri nets [91], Causal nets [6, 130], Fuzzy models [46, 47], Business Process Modeling Notation (BPMN) [88], and Event-driven Process Chains (EPCs) [109] are common formalisms in process discovery.

Process discovery can be decomposed in four possible perspectives [110]:

- **Control-flow** perspective concerns with aspects related to the process behavior [111, 112]. Examples for the control-flow perspective are all techniques that construct process models from the observed behavior described in the process data.
- **Organizational** (or resource) perspective concerns with aspects related to the resources and the organizational structure of the business process [104]. An example for the organizational perspective is a technique introduced in [114], which derives social networks from event logs.
- **Time** perspective concerns with aspects related to the timing of process events. An example for the time perspective is the *dotted chart* [103], a technique that represents process events over time.

- **Data** perspective concerns with aspects related to the properties of process instances and/or process events.[3] An example for the data perspective is the *decision miner* [95], a technique that considers the properties of process events to characterize specific choices in the process model.

Several control-flow approaches can be found in the literature. The α-algorithm [112] constructs a Workflow net by analyzing binary relations between activities.[4] Although being capable of dealing with concurrent behavior, the α-algorithm has some limitations with regard to noise, infrequent or incomplete behavior, and complex control-flow constructs (cf. Section 3.2). Several extensions to the α-algorithm have been proposed to overcome its limitations. In [4], it is presented a solution to deal with short loops. In [133], it is proposed an extension capable of dealing with non-free choice constructs. An overview of proposed extensions of the α-algorithm can be found in [124]. As an alternative to α-algorithms, the Heuristics Miner [131, 132] constructs a Causal net by computing dependency measures between activities. Being known by its capability of dealing with noise and infrequent or incomplete behavior, the Heuristics Miner is also capable of dealing with concurrent behavior as well as all complex control-flow constructs except duplicate tasks. The Genetic Miner [2, 113] is proposed to handle all common control-flow constructs and be robust to noise at once. By using genetic algorithms [38], the Genetic Miner constructs a Causal net by mimicking the process of evolution in nature on a population of candidate solutions (process models). The Genetic Miner can produce near-optimal results, but it is limited by performance issues. In [22, 23], a distributed version of the Genetic Miner is proposed to overcome this limitation. A different control-flow approach, the Fuzzy Miner [46, 47], consists of constructing hierarchical models (Fuzzy models). In this approach, the process is presented in multiple abstraction layers where less frequent behavior may be omitted or aggregated with other closely related behavior. The main limitation of the Fuzzy Miner is the incapacity of dealing with some common control-flow constructs. Based on EPCs, the Multi-Phase mining approach [121, 122] is introduced to deal with noise, infrequent or incomplete behavior, and complex split and join constructs. In this approach, the process model is built by aggregating the process instances. Based on theory of regions [14], state discovery algorithms [14, 27, 31, 32, 66, 116] construct a Petri Net from transition systems, which describe the process instances as sequences of states and the transitions between these states. Also based on theory of regions, language-based region algorithms [18, 77, 78, 119] construct a Petri Net from a language (step languages, regular languages and partial languages). The languages are defined by words representing process instances, being each word formed by a letter in an alphabet representing process events. The main limitation of the region-based approaches is the lack of generalization in the results.

Conformance Checking

Conformance consists of checking the support of a process model in the process data. This means that, given both a process model and process data, the modeled behavior of the model is compared to the observed behavior of the data in order to find similarities

[3] Properties can be defined as implicit information (i.e., information that can be derived from the process data) and explicit information (i.e., information described in the process data).

[4] A WorkFlow net (WF-net) [108] is a particular type of Petri net in which the process starts and ends in dedicated places.

and differences between them. Some examples of conformance checking can be found in the literature. In [93, 94, 96], process data are replayed on an existing Petri net in order to find misalignments between observed behavior and modeled behavior. In [6], the replaying of process data is performed on Causal nets, a formalism that captures the essential aspects of other existing formalisms used in process discovery. A concrete application of conformance checking on service-oriented systems is presented in [115].

Enhancement

Enhancement (or *extension*) consists of enriching a process model with further process information. This means that, given a process model and process data, information about different process perspectives is added to the model in order to enhance it. For instance, the model can be annotated with information about the process performance (e.g., case arrival). Some examples of enhancement can be found in the literature. In [95], process models (Petri nets) are extended with decision rules – derived from the process data – to characterize choices in the model. In [46, 47], the observed behavior described in the process data can be projected onto process models (Fuzzy models). In [131, 132], process models (Causal nets) are annotated with frequency information based on the process data.

1.3 Motivating Case Study

After introducing the research topics in the previous sections, the following case study is used to provide a motivation for the research work presented in this thesis. Conducted in the context of the DataFusion project, this case study aims at the data integration of customer experience and service data in order to enable the knowledge discovery on that data. Currently, in this case, data integration processes are defined in an *ad hoc* manner, constraining thus the scope of the data analysis. Data heterogeneity is the main issue to deal with in order to retrieve and integrate data from the existing data sources. Therefore, this case study can be seen as an abstraction of the requirements of the DataFusion project. Additionally, besides the data integration issue, the applicability of knowledge discovery techniques on the integrated data is also assessed in the case study.

1.3.1 Knowledge Discovery in Customer Experience Databases

A large global company that designs and manufactures highly exclusive quality electronic products has the customer satisfaction as one of the main performance indicators. They have been putting many efforts in acquiring the customer feedback related to their latest systems and concepts, but their insight about the customer satisfaction is not yet totally comprehensive. Although there is a large volume of data in their customer service databases, it is difficult to monitor processes like the handling of customer complaints. This happens due to data heterogeneousness, that is, the data are distributed over different databases and may be represented by different notations. Moreover, these databases have been used as independent operational systems, which means that information retrieval is not always possible.

A case study was conducted to investigate the integration of customer experience and service data. Additionally, since these data sources contain more than opera-

tional information, the application of knowledge discovery techniques (OLAP and data mining) is also considered to get insight about the customer satisfaction. Due to the similarities of this case and a data warehousing project, this case study follows the Kimball's approach [65]. As an outcome of this study, a system is designed and deployed. In this system, data can be retrieved from the distributed data sources, cleaned and transformed in such a way that knowledge discovery techniques can be applied, and finally integrated into a new data repository. For further details about this case study see [25].

Description and purpose of existing corporate data

The case study aims at a new solution toward the discovery of non-trivial knowledge from the customer service databases. Typical examples of their customer service cases are product flaws that need to be fixed, support service requests, or new functionality desires. The business managers want to know what they can learn from the (integration of) customer experience data in relation to their service quality, how they can improve their customer service process accordingly, and consequently reach a higher level of customer satisfaction. The company's customer experience databases are fed by multiple sources such as call centers, R&D, dealers, or the Internet. The collected data are typically unstructured (i.e., free text) but, at early stages of the service process, a team of specialized employees classifies the cases according to specific categories. Depending on the case, the data are used to characterize complaints or requests, which solution (or answer) can either be known or unknown. Known solutions are maintained in a knowledge management system that relies on a dedicated relational database. Unknown cases are reported to service teams to be analyzed and solved. These teams are supported by two heterogeneous relational databases, one for complaint management and the other for supporting the solution development process. The time period that each customer service case needs to be solved (i.e., from the complaint or request arrival until its solution or answer release) is defined as *solution time*. This is considered the main performance measure for which the company wants to develop a system for complaint monitoring. Such a system needs to be able to track the customer complaints along their life cycle, providing better insight about the complaint handling process. In the end, through better process performance analysis, the company will be able to improve its customer service and, consequently, the customer satisfaction.

Implemented solution

Although the solution time of a complaint can be simply defined by the time difference of when the complaint is raised and its solution is given, the procedure for computing the solution times is not as simple as it initially seems. Since the complaints follow four different stages along their life cycle, it is necessary to identify on each stage a specific complaint actually is. These stages are simply identified as *discovery*, *analysis*, *resolution*, and *release*. Let us assume that a complaint may always skip one or more stages. This happens because, for instance, a former solution may solve a recent complaint. Furthermore, other exceptional behaviors, such as cases with one missing stage, should be considered as well. Moreover, even when a complete complaint (with all of its four stages) is successfully retrieved, there may be unknown or incoherent information.

An analysis was performed to check whether the identified databases contain sufficient information about the complaints and their handling service. Combining these heterogeneous databases requires an effective data integration process. Missing information, especially foreign keys (i.e., the references that enable the relation of two different data tables), is identified as the main issue for this process. Furthermore, data normalization and transformation are also necessary to standardize or derive relevant information. The complaint life cycle information is distributed over two different databases (seven data tables). The percentage of missing (or null) information for each attribute varies up to 77%.

The implemented solution consists on a three-step data integration process: data *extraction*, *transformation*, and *loading*. First, the complaint life cycle data are extracted from the data sources to a staging area. Then, transformation services are applied to prepare the data to be analyzed according to the users requirements. This means that the complaint data need to be retrieved from the different data tables. Then, by linking the different stage data, it is possible to build the complaint case along the different stages. Eventually, data cleaning and normalization tasks are applied to treat incoherent information. Finally, the solution time can be easily calculated. The complaint case is ready to be loaded into a knowledge base.

Result analysis is achieved through OLAP techniques. The data integration process was designed in such a way that it is possible to analyze each complaint case through the combination of five different dimensions. Thus, it is possible to evaluate the customer service by either quantity (number of complaints) or quality (complaint solution time). Furthermore, different dimensions may be combined to constraint the analysis. As example, possible answers are the average of solution times for every complaint about a specific product family or the evolution of the total amount of high-priority complaints over time. *Time*, *product*, *priority*, *complaint type*, and *measurement type* are possible dimensions that can be combined with two measurements: *number of complaints* and *complaint solution time*.

Conclusions

This case study focused on the design and development of a system capable of performing data integration and knowledge discovery on process data. It was demonstrated that, through combining customer experience information, it is possible to obtain a better overview about customer service processes. This actionable knowledge may be used at both strategic and operational level. Once that the system's knowledge base relies on coherent and integrated data, this new knowledge source outstands its own data sources in terms of data trustworthiness and readiness. So far, OLAP-based techniques were applied to exploit the business process information through different perspectives. However, other issues such as improving the detection of bottlenecks and using the data in NPD projects are not solved. Table 1.1 summarizes the specific problems exhibited before and after the application of the implemented solution.

As identified in Table 1.1, the main limitation of the implemented solution is the impossibility of determining the source of long throughput times. This issue can be generalized to the system's incapacity of performing process discovery and analysis. Although a concrete business process is described in the data, the proposed solution simply focuses in four predetermined process stages. This high-level of abstraction is the ideal to assess the process performance of functional units, but it does not provide

	Specific problem (requirements)	*Implemented solution*
Before the application of the new solution	1. Time to solution difficult to retrieve	Data integration process
	2. Time to solution difficult to analyze	Knowledge discovery techniques
After the application of the new solution	1. Evaluating customer services by quality and quantity (calculate time to solution)	Problem partially solved
	2. System still not effective to detect the source of long throughput times	Problem to be solved (process mining)
	3. Data not used in NPD (potential usage)	Problem to be solved (knowledge management)

Table 1.1: Overview of the case study results.

any insight into the root causes of specific issues (e.g., bottlenecks). Therefore, the integration of process mining in the solution is required to overcome this limitation. By applying process discovery techniques on different business perspectives, it will be possible to discovery and analyze the process at different levels of abstraction and according to multiple dimensions (e.g., the process of handling high-priority complaints).

1.4 Multidimensional Process Mining

By applying process mining techniques on process data, it is possible to discover the business process as it was executed. However, so far, these techniques are typically designed to focus on specific aspects of the process, omitting information potentially relevant for the process comprehension. Eventually, this limitation may be overcome by some data preprocessing (e.g., redefining aspects of the process or filtering the data), though in a laborious and non-integrated way.

The dynamic integration of business process dimensions into process mining can bring the process analysis to another level. Considering the attributes of the event log as dimensions, process analysis can be performed in such a way that events and workflow can be constrained by specific process information. Traditionally, only a single dimension (or a static set of dimensions) is considered. For example, the *activity* is typically the only considered dimension in control-flow mining for deriving process models. Additionally, social network analysis techniques basically consider the *resource* dimension. However, neither the resources can be described in process models nor the activities in social networks. In a multidimensional approach, process analysis can be dynamically constrained by any dimension or combination of dimensions the analyst considers relevant.

In this thesis, a multidimensional approach is introduced to facilitate the dynamic integration of business process dimensions into process mining. Basically, this approach aims at knowledge discovery on process data by combining concepts from process mining, OLAP, and data mining. Although the approach is designed using the requirements of the DataFusion project, it is more general in its basic nature. Hence, it can be applied in other domains (i.e., NPD is one possible application where the multidimensional approach can be applied).

The multidimensional approach consists of a sequence of steps in which process

data are transformed first into process information, and then into process knowledge. An initial overview of a multidimensional approach for process discovery is provided in Figure 1.3. All components identified in this overview will be discussed in the core chapters of this thesis. Furthermore, after introducing and discussing the components, updated overviews of the multidimensional process discovery approach will be provided along the thesis.

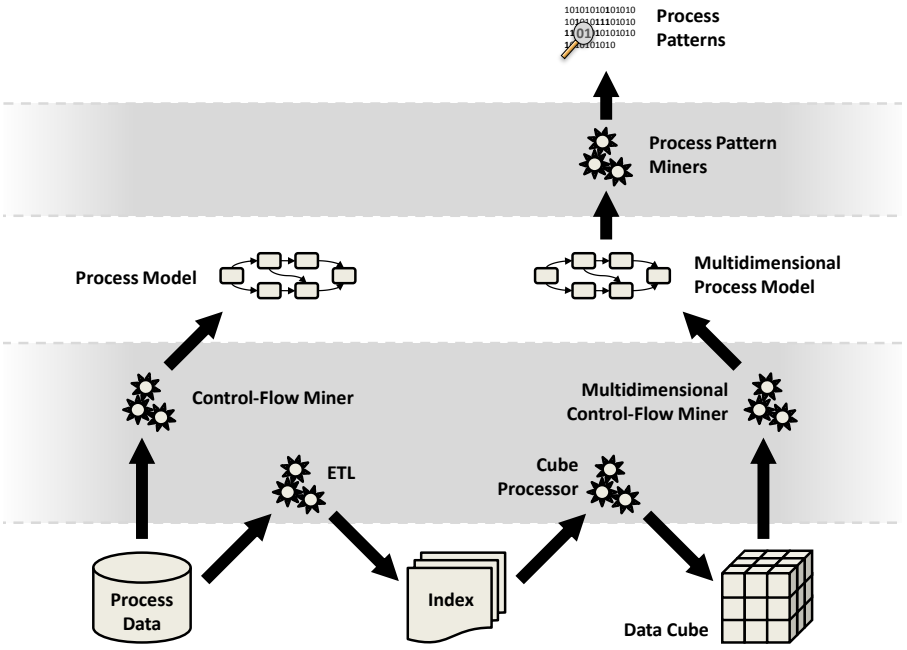


Figure 1.3: Initial overview of the multidimensional process discovery approach. Layers in gray color characterize data transformations, while the others identify process data or results. Arrows represent the data flow.

Both traditional and multidimensional process discovery approaches are represented in Figure 1.3. These approaches can be described as follows.

Traditional process discovery approach consists of applying current process discovery techniques directly on process data. This approach is represented by the leftmost objects in Figure 1.3 (*Process Data*, *Control-Flow Miner*, and *Process Model*). Although only the *control-flow* perspective of process discovery is identified in the overview diagram, other perspectives are assumed to be part of the traditional approach as well.

Multidimensional process discovery approach consists of

i. indexing the process data for optimizing the data retrieval,

ii. computing a data cube for enabling the dynamic exploitation of multiple aspects (dimensions) described in the process data,

iii. performing process discovery on a specific business perspective (i.e., a partition of the process data characterized by a set of dimensions), and

iv. eventually, extracting non-trivial patterns from multidimensional process models by applying data mining-based techniques.

This approach is represented by all objects in Figure 1.3 except the *Control-Flow Miner* and the *Process Model*. Unlike traditional process models, multidimensional process models can be used to combine the *control-flow* perspective of process discovery with the other perspectives (i.e., *time*, *resource*, and *data*).

Related Work

As mentioned before, process mining techniques provide strategic insight into the business processes. Process discovery, conformance checking, and enhancement are possible applications [106, 110, 118]. However, the state-of-the-art process mining techniques do not support yet multidimensional analysis. Nevertheless, some research has been done in multidimensional process analysis [83, 86]. Workflow ART is a proposed framework to explore the three main business process dimensions *Action* (or *Activity*), *Resource* and *Time*. Although several predefined business measures can be analyzed, this approach is not as flexible as an OLAP-based technique. A fully multidimensional approach for business process analysis can be found in [83]. Relying in the multidimensional data model, this approach maps the different aspects of the business process into a data cube. However, only numerical information is considered.

The multidimensional process discovery approach applies OLAP – and data mining – concepts on business process data. This means that predefined business measures can be exploited under any combination of dimensions, producing multidimensional views of the process data. Therefore, the process visualization plays an important role in multidimensional process discovery and analysis. In [62, 100], different visualizations are proposed for different types of business process analyses. In [19], the extension of traditional model representations with resource-related information is proposed.

1.5 Research Questions

In this section, we define two research questions to be investigated in this thesis. Covering the enterprise information systems and information retrieval domains, a preliminary research question can be stated as follows.

(Q') *How to establish an effective connection between the – operational – data sources and the knowledge discovery techniques?*

By answering this preliminary research question, it is possible to position the main focus of this thesis (the knowledge discovery on process data) with respect to the main goal of the DataFusion project (the integration of information from the existing NPD datasets).

Covering the process mining, OLAP, and data mining domains, the main research question of this thesis aims at the discovery and analysis of business processes by integrating dynamically the different aspects (dimensions) of the process. This question is stated as follows.

(Q) *What is the impact – and the benefit – of the dynamic multidimensional discovery and analysis of business processes?*

This research question can be decomposed into six sub-questions.

(q_1) *Which traditional process mining techniques can be extended with multidimensional capacity (i.e., be applied using different dimensions, by turning traditional analyses into multidimensional)?*

(q_2) *How can traditional process discovery be extended with multidimensional capacity?*

(q_3) *What kind of knowledge can be achieved with dynamic multidimensional process discovery and analysis?*

(q_4) *How can multidimensional process discovery results be exploited for knowledge discovery?*

(q_5) *What kind of non-trivial patterns can be – automatically – derived from multidimensional process discovery results?*

(q_6) *What is the added value of dynamic multidimensional process discovery and analysis to the stakeholders?*

Figure 1.4 positions the research question Q and sub-questions q_{1-6} with respect to the process mining, OLAP, and data mining domains. Multidimensional process discovery can be performed either by hypothesis-driven analysis (OLAP-based approach) or by the extraction of non-trivial patterns (data mining-based approach). Sub-questions q_1 and q_2 aim at the extension of the traditional process mining techniques with multidimensional capacity by combining the process mining and the OLAP approaches. Sub-questions q_3 and q_4 concern the study of the potential of dynamic multidimensional process discovery. Sub-question q_5 aims at the application of the data mining-based approach on multidimensional process discovery results. Finally, the impact (added value) of the proposed multidimensional approach on real-life applications is evaluated by answering the last sub-question (q_6).

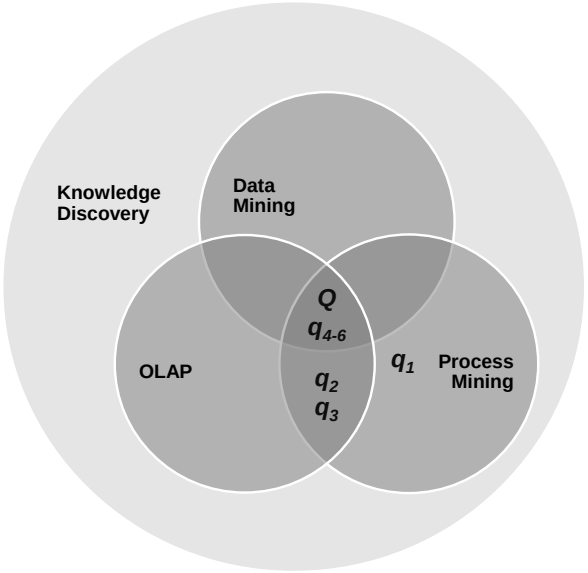


Figure 1.4: Positioning of the research questions Q and q_{1-6} with respect to the process mining, OLAP, and data mining domains.

Figure 1.5 positions the research question Q and sub-questions q_{1-6} with respect to the multidimensional process discovery approach.

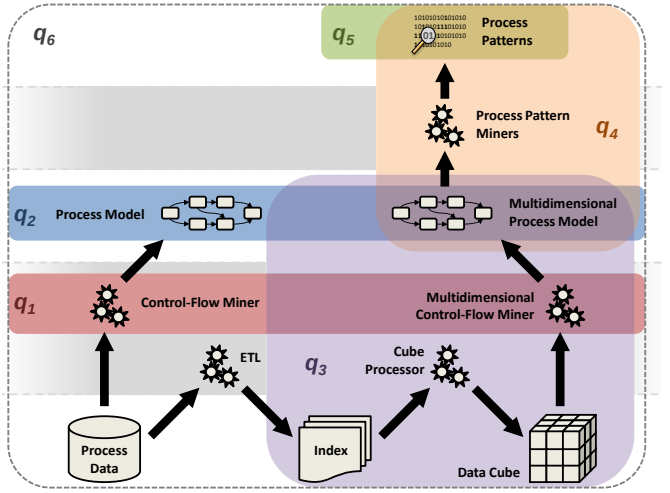


Figure 1.5: Positioning of the research questions q_{1-6} with respect to the multidimensional process discovery approach.

1.6 Outline

The structure of this thesis can be described as follows.

Chapter 1, the current chapter, firstly introduces the DataFusion project and the general domain of NPD and knowledge discovery. Then, a case study is used to motivate the need for multidimensional discovery and analysis of business processes. A partial answer for research question q_1 is provided in this chapter.

Chapter 2 introduces the concept architecture of a system in which the requirements of the DataFusion project are fulfilled. This concept architecture can be considered as the answer for research question Q'.

Chapter 3 presents a methodology to represent and construct multidimensional process models from process data. Research questions q_1 and q_2 are addressed in this chapter.

Chapter 4 discusses the similarity aspects of multidimensional process models. In this chapter, similarity functions are defined to facilitate the comparison between elements of multidimensional process models. Research question q_2 is further addressed in this chapter.

Chapter 5 introduces an OLAP-based framework for process discovery, and describes how the multidimensional discovery and analysis of business processes can be achieved. Research questions q_3 and q_4 are addressed in this chapter.

Chapter 6 describes the application of data mining functionalities on the multidimensional process discovery approach. Research question q_5 is addressed in this chapter.

Chapter 7 presents a quantitative and qualitative evaluation of the multidimensional process discovery approach described in this thesis. This evaluation provides the answer for research question q_6.

Chapter 8 concludes this thesis by providing a final discussion as well as contributions, applications, limitations, and future work. Research question Q is discussed in this chapter.

Appendix consists of an alternative approach for building multidimensional process models from an Event Cube (concept defined in Chapter 5), relying on a reference process model.

In Figure 1.6, it is provided the positioning of the core chapters (i.e., chapters $2-7$) with respect to the multidimensional process discovery approach.

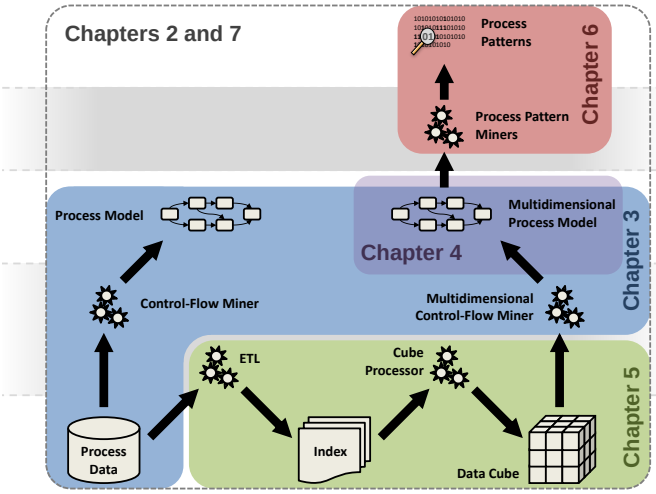


Figure 1.6: Positioning of the core chapters with respect to the multidimensional process discovery approach.

Chapter 2

Concept Architecture

The term architecture is assumed to be a specification of a system, and can be defined as follows [44]:

> *The architecture of a (corporate) information system defines that system in terms of functional components and interactions between those components, from the viewpoint of specific aspects of that system, possibly organized into multiple levels, and based on specific structuring principles.*

In this chapter, we introduce the architecture of a (corporate) information system in which the requirements of the DataFusion project are fulfilled (i.e., a concept system to establish an effective connection between the data sources and the knowledge discovery techniques). This architecture is used to position the multidimensional process discovery approach presented in this thesis (chapters $3-6$) in the context of the DataFusion project.

As described in Section 1.1, the DataFusion project aims at (i) the extraction and integration of operational data from the existing NPD operational databases, (ii) the transformation of NPD operational data into structural, richer and prioritized field feedback information, and (iii) the knowledge discovery on the field feedback information in order to identify non-trivial patterns. Therefore, three distinct functional sub-areas can be clearly identified: *data sources* (i.e., the operational databases), *data preparation* (i.e., the sub-system that transforms data into information), and *data analysis* (i.e., the tools for data analysis and knowledge discovery). Logically, the data flow goes from the data sources to the data analysis (Figure 2.1).

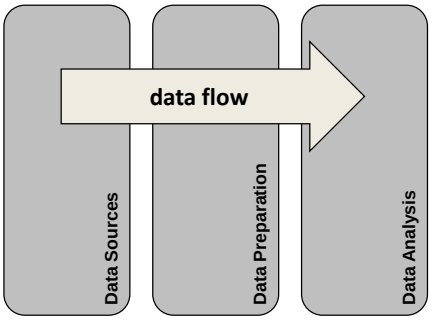


Figure 2.1: The three functional sub-areas of the concept architecture.

Taking the three functional sub-areas of the concept architecture as a starting point, we first describe the project's reference architecture (Section 2.1) in order to identify the main functional components and their interactions. This reference architecture is intended to be an abstract blueprint of the project's information system with business-oriented specifications. Next, we instantiate two alternative architectures (sections 2.2 and 2.3) from the reference architecture, providing more technology-oriented specifications and concrete component descriptions. The first one, a data warehousing system, can be seen as the logical option once that this kind of system is optimized to integrate data from heterogeneous sources for decision supporting. However, since NPD processes are dynamic and multidisciplinary, some of the prerequisites for implementing a data warehousing system may not be met. Hence, as an alternative approach, the search engine systems are considered for overcoming these issues. Finally, in Section 2.4, we introduce the architecture of a framework in which knowledge discovery can be performed on the field feedback information, fulfilling thus both the third objective of the DataFusion project and the requirements of the research work presented in this thesis. For all the architectures, three main dimensions are considered to describe the characteristics of architecture models [44]:

The abstraction dimension describes the detail level of the architecture description in terms of component characterization. The domain of this dimension can be defined by four distinct abstraction levels. At the first and highest level, the architecture is represented in terms of *class type components*. At the second level, *system type components* can be used to represent the architecture. At the third level, the architecture is represented by *vendor type components*. Finally, at the last and lowest level, *vendor version components* provide a highly detailed representation of the architecture.

The aggregation dimension describes the detail level of the architecture description in terms of component representation. At one end of this dimension domain, the architecture of an interorganizational information system covers a wide range of information systems. At the other end, an architectural submodule covers specific functionalities by representing functional elements.

The realization dimension describes the orientation of the architecture description in terms of component specification. At one end of this dimension domain, business-oriented specifications focus on the motivation for developing the information system described in the architecture. At the other end, technology-oriented specifications provide the technical requirements for the architecture realization.

The detail level of the descriptions of the DataFusion project's functional sub-areas (Figure 2.1) is characterized in Figure 2.2 according to these three dimensions.

Remark that the focus of this chapter is simply to introduce an architecture that supports the DataFusion project needs. Hence, there are dimensions and architectural aspects (i.e., specific characteristics on which the architecture focuses) that are not considered. Examples of other dimensions can be found in [43]. For an overview of architectural aspects see [44].

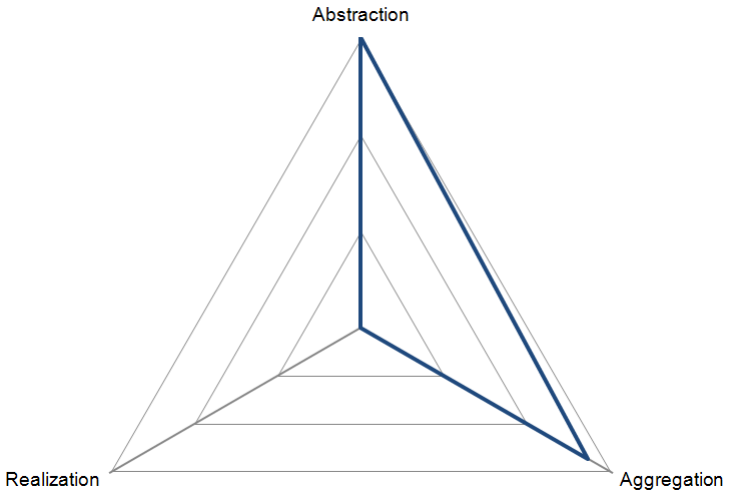


Figure 2.2: Detail level of the functional sub-areas descriptions.

2.1 Reference Architecture

The term – software – reference architecture can be defined as follows [15]:

> *A reference architecture consists of a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them.*[1]

Basing on a framework for analysis and design of reference architectures introduced in [12], we first characterize the reference architecture of the DataFusion project in terms of type. This characterization provides insight into the objective, the context, and the design of the reference architecture. The goal of the project's reference architecture is the standardization of concrete architectures (e.g., the data warehousing system and the enterprise search engine architectures, presented in the next sections). The context can be decomposed in three parts: (i) where and (ii) when the architecture will be used, and (iii) who defines it. Although it can be applied in a broader scope, the DataFusion architecture is meant to work in a single organization. Since some of its components are not yet developed (by the time of its design), the project's reference architecture is considered as preliminary. Research centers and user groups (i.e., the members of the DataFusion project) are the stakeholders involved in the design of the reference architecture. The design process consists of describing the architecture's components and interfaces at a high level of aggregation. Architecture details are provided by semi-formal specifications of semi-concrete elements. This generic architecture representation facilitates the architecture's components applicability in different contexts. The characteristics of the reference architecture of the DataFusion project are summarized in Table 2.1.

The reference architecture of the DataFusion project consists of the general design of a structure for the information system that fulfills the project's requirements. In this architecture, the three functional sub-areas previously identified are considered: data sources, data preparation, and data analysis. These sub-areas can be seen as class type

[1] *A reference model is a division of functionality together with data flow between the pieces* [15].

Aspect	*Description*	*Value*
Goal	Why is it defined?	Standardization
Context	Where will it be used?	Single Organization
Context	When is it defined?	Preliminary reference architecture
Context	Who defines it?	Research centers and user groups
Design	What is described?	Components and interfaces
Design	How detailed is it described?	Aggregated components
Design	How concrete is it described?	Semi-concrete elements
Design	How is it represented	Semi-formal element specifications

Table 2.1: Characterization of the reference architecture in terms of type (according to [12]).

components (abstraction dimension). Also, the architecture describes a wide range of information systems (aggregation dimension) by using business-oriented descriptions (realization dimension). By moving one level down on both abstraction and aggregation dimensions, we intend to provide further details about system type components and their interactions. By moving one level up on the realization dimension, we intend to describe some technology-oriented specifications. These operations are summarized in Figure 2.3 in which the different line styles represent the detail level of the reference architecture (solid line) and the functional sub-areas (dashed line) descriptions.

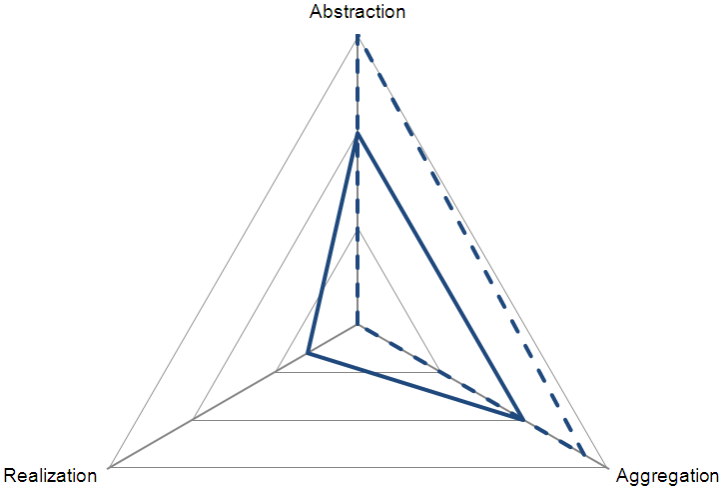


Figure 2.3: Detail level of the reference architecture description.

The main focus of this reference architecture is the data flow, i.e., the transformation of data into information and of information into knowledge. Figure 2.4 presents an overview of the reference architecture by combining the component-oriented with the columned architecture styles [44]. The three functional sub-areas are identified at the top. In each column, the corresponding functional sub-area is decomposed in one or more system type components (abstraction dimension). The data flow through the different components is represented by the arrows. Remark that, to keep simplicity, the – multiple – data sources are considered both functional sub-area and generic system type.

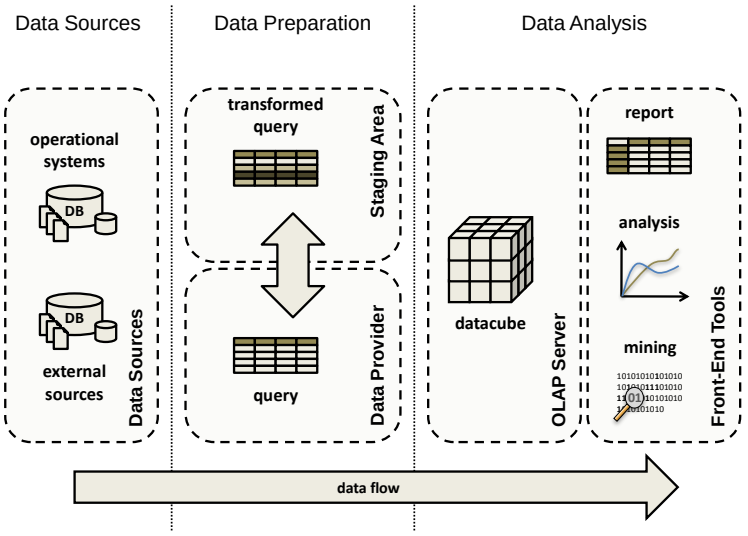


Figure 2.4: Overview of the reference architecture.

According to the reference architecture in Figure 2.4, there are five system type components:

Data Sources refer to all types of accessible systems (or data structures) from which data are obtained. Databases, data streams, and computer files are typical examples of data sources.

Staging Area encloses all services that transform and prepare data to be exploited. Filtering, cleaning, normalizing, or deriving new information are some of the possible operations that can be performed in the staging area.

Data Provider refers to systems that facilitate the integration of data from heterogeneous data sources, and make it accessible for exploitation. Data warehousing systems and search engines can be considered as data providers.

OLAP Server is a data manipulation engine for dynamic multidimensional analysis of consolidated enterprise data. Relying on the multidimensional data model, OLAP tools organize the data in such a way that it is possible to have multiple perspectives of the data in the same analysis.

Front-End Tools consist of all sorts of techniques for data exploitation. Knowledge discovery techniques such as data mining and OLAP are typical applications for data analysis.

Remark that the staging area and the data provider are often considered as different parts of the same system. Also, it is assumed that each system maintains a metadata repository about its own objects. The metadata describes the objects in terms of structure, meaning, and relationships. Additionally, information about the system's performance, history, and processes may be provided as well.

In order to explain the data flow through the system type components, we provide in Table 2.2 the characterization of the component's data objects. The data sources

can be seen as a set of databases from which – operational – data can be selected and extracted. After extraction data consists of a dataset, which is characterized by given query conditions and maintained in a data provider. Eventually, in order to prepare the data for analysis, the extracted dataset is submitted to some data transformation services in the staging area for consolidation. Next, the data are loaded into an OLAP server for materialization, which means that the data is organized in a data cube in order to make it possible to perform the data exploitation through multiple perspectives. Finally, the data can be exploited using the front-end tools.

Component	*Object*	*Type*
Data Sources	Operational Data	Database
Data Sources	External Data	Dataset
Data Provider	Query	Dataset
Staging Area	Transformed Query	Dataset
OLAP Server	Data Cube	Database
Front-End Tools	Report	Result
Front-End Tools	Analysis	Result
Front-End Tools	Mining	Result

Table 2.2: Reference architecture: characterization of the component's data objects.

In the next two sections, we decompose the reference architecture in terms of vendor type components (abstraction dimension), corresponding functional modules (aggregation dimension), and technology-oriented specifications (realization dimension). Two different approaches are considered as data provider: the data warehousing system and the enterprise search engine. The first one can be seen as the logical option once that this kind of system is optimized to integrate data from heterogeneous sources for decision supporting. However, since NPD processes are dynamic and multidisciplinary, some of the prerequisites for implementing a data warehousing system may not be met. Hence, as an alternative approach, the search engine systems are considered for overcoming these issues. For both approaches, the detail level of the architectures descriptions is summarized in Figure 2.5. This time the solid line represents both the data warehousing system and the enterprise search engine architectures, while the dashed line refers to the reference architecture.

2.2 Data Warehousing System

A data warehousing system (DWS) consists of the technology that supports the integration of data from heterogeneous sources, and is typically designed for decision making. Mainly from operational systems, data are extracted from the sources, transformed according to specific requirements, and loaded into a single data repository called *data warehouse*. The data in the data warehouse comprise the following characteristics [59]:

- *Multidimensional*: The system's data can be exploited and analyzed combining different dimensions.
- *Subject-oriented*: The system's data are organized so that all data elements relating to the same real-world event or object are linked together.

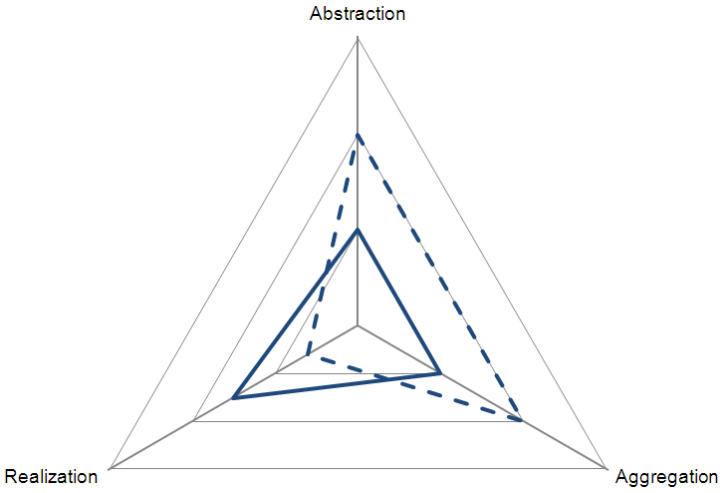


Figure 2.5: Detail level of the descriptions of the data warehousing system and the enterprise search engine architectures.

- *Time-variant*: The changes to the sources' data are tracked and recorded so that reports can be produced showing changes over time.
- *Non-volatile*: The system's data are never over-written or deleted – once committed, the data are static, read-only, and retained for future reporting.
- *Integrated*: The data warehouse contains data from most (or all) of the organization's operational systems and these data are made consistent.

A data warehousing system is autonomous from the data sources, and does not contain – exactly – the same information as the operational databases (i.e., the system also maintains summarized data and historical information). Thus, the data warehouse's data are always available even when the operational systems are not.

Figure 2.6 presents the component diagram of a data warehousing system by combining the component-oriented with the layered architecture styles [44]. The diagram describes a generic data warehouse architecture with staging area and data marts, which can support the most predominant data warehouse reference architectures [128]:

Hub and Spoke Architecture (also known as *Corporate Information Factory*) consists of a data warehousing system with a normalized data warehouse and dependent data marts. Based on the data-driven approach (i.e., the data warehousing system is designed according to the existing data in the operational source systems), this architecture was proposed by Inmon [60] and is especially designed to deal with scalability issues. The basic idea is that atomic level data are stored in a normalized data warehouse (i.e., a third normal form (3NF) relational database). This normalized data warehouse is used as the data source of dependent data marts or data analysis tools. The dependent data marts are oriented to specific purposes, departments and users, and contain dimensionally structured summary data.

Kimball Bus Architecture consists of a data warehousing system with a centralized data warehouse or linked data marts. Based on the goal-driven approach (i.e.,

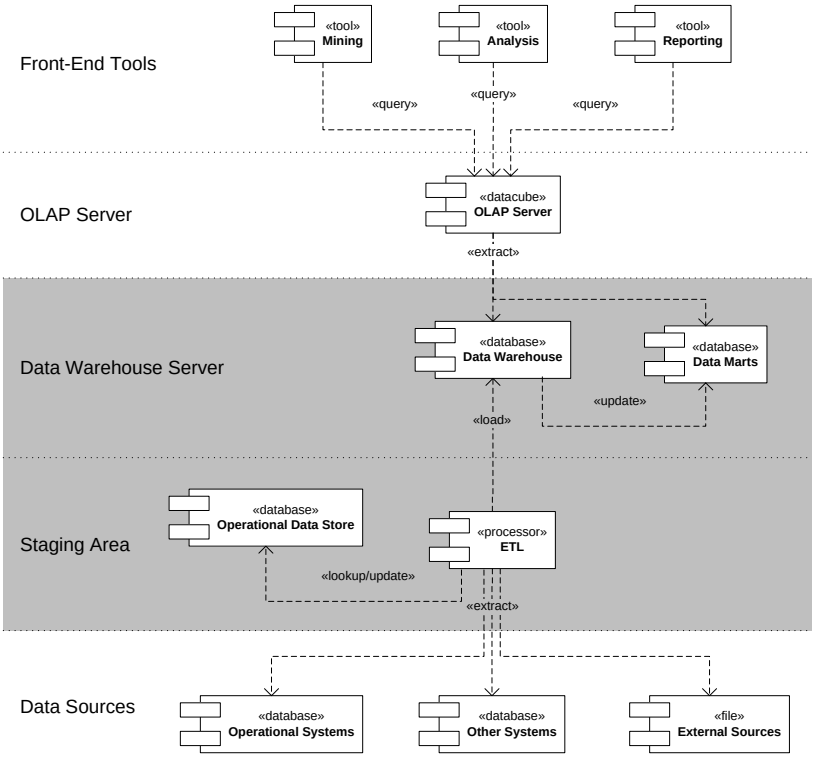


Figure 2.6: Component diagram of a data warehousing system. The gray-colored layers identify core components of the system.

data warehousing system is designed according to specific strategic goals), this architecture was proposed by Kimball [65] and is especially designed to deal with performance issues. The basic idea is that, in order to provide a dimensional view of the data, both atomic and summarized data are organized under the multidimensional data model (Section 5.1) in either the data warehouse or the data marts.

The data warehousing system's component diagram (Figure 2.6) is a specialization of the reference architecture's overview diagram (Figure 2.4). Therefore, the system type components are decomposed in one or more vendor types components. The connectors, the lines between the components, represent how the data flows from one component to another. Note that the component's interfaces are considered implicit in the components.

According to the component diagram in Figure 2.6, a data warehousing system relies on four main components:

ETL is a predefined process that involves (i) extracting data from some data sources, (ii) transforming the extracted data according to specific requirements, and (iii) loading the transformed data into some data repository. In a data warehousing system, the ETL is executed in a regular basis to populate – and keep updated

– the data warehouse.

Operational Data Store (ODS) is an auxiliary repository that supports ETL processes. An ODS maintains the necessary information for the ETL's operations such as surrogate keys and input data. For example, financial information may need to be converted into a specific currency. So, as input data, exchange rates are necessary for the conversion, but not relevant for analysis.

Data Warehouse is a repository that maintains historical information under a unified schema.

Data Mart is a subset of the data warehouse, which is oriented to specific subject, requirements, and users.

One of the advantages of a data warehousing system is its capability for performing on-the-fly integrated dimensional analyses on enterprise data. Using complex queries in user-friendly tools, the users are provided with a broad dimensional view of the data. Additionally, the ETL processes ensure data quality and consistency, characteristics that are difficult to achieve using data directly from the operational systems.[2] Nevertheless, these characteristics are achieved under an important assumption: data sources remain static over time (i.e., the structure of the data sources should not change). This happens because the data warehousing system relies on predefined data integration processes in which the data definition (meaning and structure) must be preserved on pain of information consistency loss. Since NPD processes are dynamic (i.e., the way they are executed may vary according several factors), it is not likely that the existing NPD operational databases remain static over time. Furthermore, different NPD projects lead tendentiously to the creation of new data repositories [28].

A data warehousing system is optimized to handle factual data, expressed typically by numerical values. In contrast, the DataFusion project focuses essentially on free-text data. Additionally, the data warehousing systems rely on very structured data schemas that imply well-defined facts. Since NPD environments have a multidisciplinary nature, the fact definition from free-text data is a major issue because each disciplinary group has its own perspective on reality. Hence, the distinct handling of all of these perspectives is a utopian task as well as their combination into a generic single case. This happens because a text can have multiple interpretations and its generalization process would vanish the data roots, restricting the data exploitation to the existing information in the data warehousing system.

Using the motivation described above, we propose another approach for the DataFusion project. The main idea is to combine the traditional data warehousing system with the web search engines. The search engines are mainly based on information retrieval algorithms, which work effectively well on free-text. Furthermore, these searching tools are designed to work on dynamic environments such as the web. Thus, if one is able to migrate these abilities to the data warehousing system, the power of the developing NPD data warehouse will be optimized to this particular environment. To do so, a different approach is considered: the enterprise search engine.

[2]Remark that achieving data quality and consistency depends on the effectiveness of the ETL processes as well as on the data in the operational systems.

2.3 Enterprise Search Engine

The enterprise search engine consists of the technology that searches data from heterogeneous sources within an enterprise [53]. Like in a traditional web search engine, all the available data are fetched from the sources, analyzed at a content level, normalized to given standards, and indexed into the system's index. Given a query, specific data may then be located and retrieved from the sources. Figure 2.7 presents the component diagram of the enterprise search engine system by combining the component-oriented with the layered architecture styles. Like the data warehousing system's component diagram, this component diagram is a specialization of the reference architecture's overview diagram (Figure 2.4). The components represent vendor types components, while the connectors describe how the data flows from one component to another. The component's interfaces are considered implicit in the components.

According to the component diagram in Figure 2.7, the enterprise search engine relies on eight main components:

Crawler is a software agent that searches automatically the data sources in order to discover the current state of the sources' data. Like a web crawler that browses the web, a crawler should be capable of identifying changes on the data sources over time, indexing and updating the available data into the enterprise search engine's index.

Metadata is an additional metadata repository that describes the sources' data in terms of structure, meaning, and relationships. These data can be used to facilitate the data source discovery and the data indexation. Eventually, this repository can support the execution of some data transformation services.

Ontology defines formally concepts and entities within a domain, and the relationships among them.

Index is a data structure that facilitates the data retrieval from the data sources. Like a book's index, these indices map keywords to the set of data structures (in the data sources) containing the keyword.

Query Processor is a computer program that interprets the users input (given through the user interface) and processes it by searching the index. The output (i.e., the results) consists of the list of entries that identify where the data satisfying the input conditions appear.

Transformation Services component is a computer program that transforms data according to specific requirements. Data transformations consist of applying one or more transformation functions on retrieved data from operational systems or the knowledge base. Eventually, metadata and ontology information is considered to support the data transformation.

Knowledge Base is an extra repository to maintain results the users consider relevant to archive. These results may be data directly retrieved from the data sources, information obtained from the transformation services, or data analysis results. Eventually, this repository may be used for knowledge sharing within the organization.

User Interface is a computer program that allows the users to (i) search and retrieve data from the operational systems (or the knowledge base), (ii) transform data using transformation services, (iii) make data available for materialization and data analysis, and (iv) store results in the knowledge base.

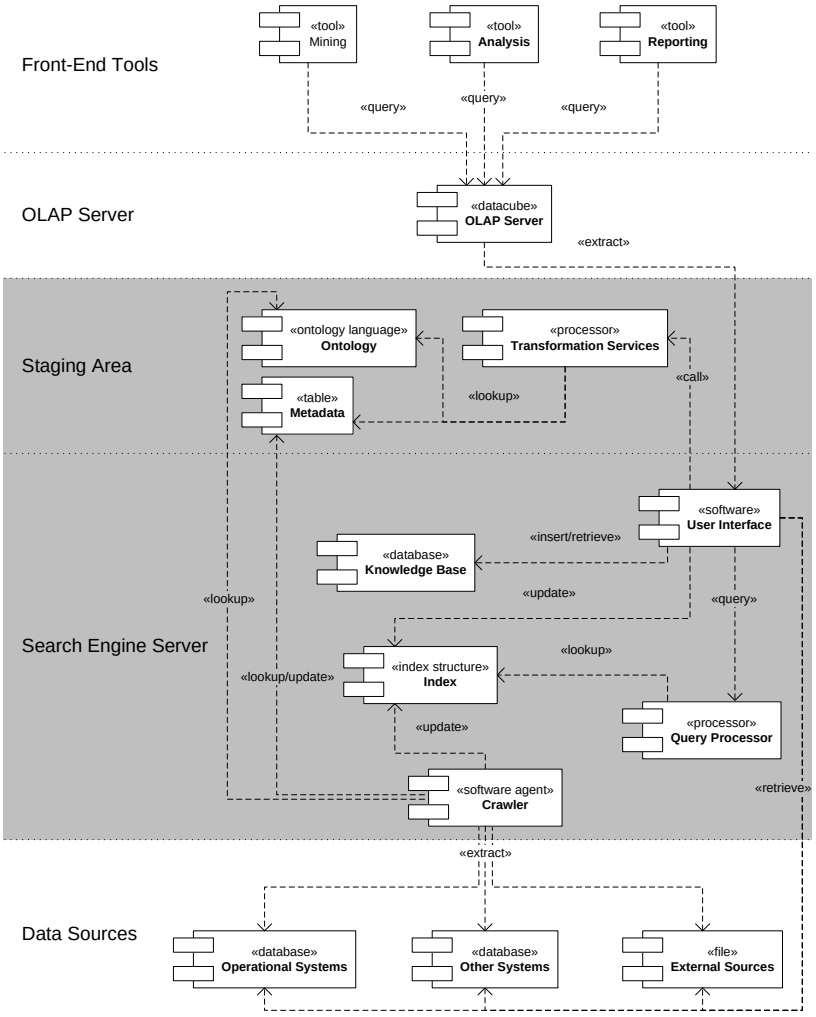


Figure 2.7: Component diagram of the enterprise search engine. The gray-colored layers identify core components of the system.

Unlike the data warehousing systems, the enterprise search engine is dependent on the data sources. All the accessible data of the operational databases are simply indexed in the system, being eventually retrieved from the sources. Hence, the data redundancy is minimized to the detriment of query performance. Any sort of data materialization must thus be achieved after querying by applying some transformation services (staging area) on the retrieved data. This explains the positioning, in the component diagram, of the staging area between the search engine and the OLAP server. Eventually, results are inserted in the system's knowledge base and made searchable for later retrieval (i.e., the system's index is updated). Obviously, the knowledge base is autonomous from the data sources and may contain redundant data.

In order to assess the differences between the data warehousing system and the

enterprise search engine, we first describe for this architecture the five characteristics that define the main requirements of the data warehousing systems. Next, in Table 2.3, we compare the characteristics of both systems as well as some other aspects mentioned before. Then, we identify the major issues and challenges of using the enterprise search engine as the DataFusion architecture.

- *Multidimensional*: Like the data warehousing systems, the data can be exploited and analyzed combining different dimensions.
- *Orientation*: Since the sources' data are simply indexed, there is no specific orientation for the enterprise search engine systems. The same applies when some results are archived in the knowledge base once that this repository can maintain any sort of information (i.e., it is not necessary to exist any relation among the data elements).
- *Time-variant*: The changes to the sources' data are tracked, but only for keeping the system's index up-to-date. This means that reports cannot be produced showing changes over time. This issue can eventually be overcome if the changes are saved in the knowledge base.
- *Non-volatile*: The system may produce different results over time for the same query, which leads to inconsistencies in the query results. This happens because the sources' data can be over-written (or deleted) at any time. Hence, even that the knowledge base's data are made read-only and not deletable, query results – depending on the data sources – may vary over time. Tracking the changes on the data sources and saving them in the knowledge base may solve partially this issue.
- *Integrated*: The enterprise search engine may maintain, in the knowledge base, data from most (or all) of the organization's operational systems. However, these data may not be completely integrated or even consistent if the integration processes are not capable of dealing with complex or nonexistent data relationships.

Characteristics	Data Warehousing System	Enterprise Search Engine
Multidimensional	Yes	Yes
Orientation	Subject	Generic
Time-variant	Yes	No
Non-volatile	Yes	No
Integrated	Yes	No/Partially
Data sources dependency	No (except for updating)	Yes
Query performance	Optimal	Limited
Data materialization	Before querying	After querying
Data redundancy	Maximal	Minimal
Focus	Localized	Widespread

Table 2.3: Comparison between the traditional data warehousing system and the enterprise search engine.

Although both systems can act as data integrators and providers, Table 2.3 demonstrates that data warehousing systems and the enterprise search engine are divergent in many aspects. This happens because both systems are designed for different purposes. On the one hand, a data warehousing system focuses in specific – and limited

– operational data for simply transforming them into strategic information. On the other hand, the enterprise search engine takes into account all the accessible data, and makes them available for generic purposes. By applying some transformation services on the search engine results, data can also be transformed into strategic information, but in a much more flexible manner and a broader scope. However, there are some issues to overcome.

The enterprise search engine may not be time-variant and non-volatile, and the results provided by this system may not be completely integrated. As mentioned before, the knowledge base can be part of the solution for some of these issues. Additionally, intelligent transformation services can enhance the integration processes as well as the data quality. Nonetheless, a further analysis on these issues is required in order to assess the feasibility of deploying the enterprise search engine on NPD operational systems for knowledge discovery.

2.4 Multidimensional Process Explorer

In this section, we introduce a possible instantiation of the enterprise search engine architecture in terms of version components (abstraction dimension), corresponding functional elements (aggregation dimension), and concrete technology-oriented specifications (realization dimension). The detail level of this architecture description is summarized in Figure 2.8 in which the different line styles represent the detail level of the instantiation (solid line) and the enterprise search engine (dashed line) architectures.

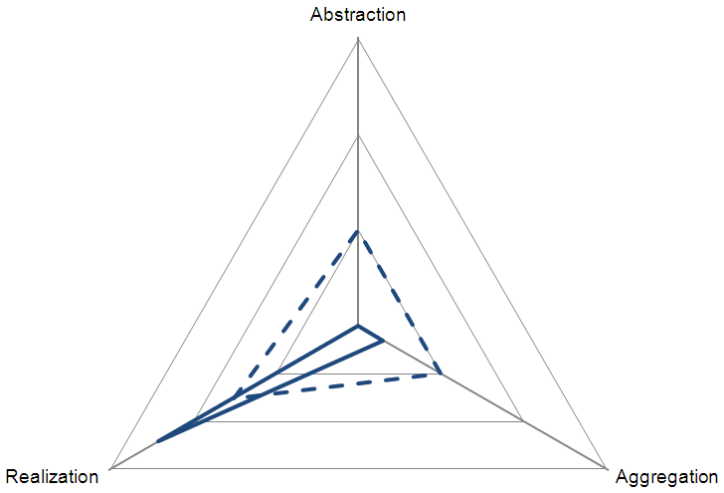


Figure 2.8: Detail level of the descriptions of the Multidimensional Process Explorer architecture.

This instantiation refers to the Multidimensional Process Explorer framework, a process mining tool for process discovery and analysis. As input, process data (i.e., information about the execution of business processes that can be used to replay the executed events, identifying details such as originators or temporal references) are read from either event logs (computer files) or data streams. As output, multidimensional

process models (i.e., graphical representations of business processes) and related analysis results are presented to the user through the framework's graphical user interface.

This instantiation consists of three distinct components. First, the event logs are considered as instance of the external sources. Second, the Enterprise System and Analysis Tool (ESAT) system is instantiated as data provider. The ESAT is an Apache Solr-based enterprise engine tool [70], which can be used to index relational databases as well as document files such as PDF, Word or Excel.[3] This system also offers an intuitive graphical user interface in which the user can select the right data to be retrieved and analyzed. Third, the Multidimensional Process Explorer framework is identified as instance of both OLAP server and analysis tools. This framework is a process mining technique that organizes the event data in such a way that it is possible to exploit the different business dimensions [92]. This means that the process discovery and analysis can be performed considering multiple dimensions and measures of interest. A data cube of events is defined to support these multidimensional analyses, which can either be done directly on the cube or using the cube as basis for process mining techniques such as the Flexible Heuristics Miner [130]. Implementing the main OLAP operators (cf. Section 5.1), the proposed framework provides to the business analyst the opportunity to drill-down the process information without complex queries and long response times.

Figure 2.9 presents an overview of the Multidimensional Process Explorer framework by combining the component-oriented with the columned architecture styles. Focusing on the data flow, this overview shows the component's data objects and their role in the execution of the framework. The data flow through the different components is represented by the arrows. Remark that this framework conforms to the reference architecture described in Section 2.1. In this overview, the staging area is omitted as well as the data flow from the data sources to the data provider. As long the process cases are recognized as so, the data provider component may be either a data warehousing system or an enterprise search engine.

In order to explain the data flow through the components, we provide in Table 2.4 the characterization of the component's data objects. The Multidimensional Process Explorer framework relies on process data to build a data cube of events. These data can be provided by either an event log or a data connection to a data provider (i.e., a data warehousing system or an enterprise search engine). The data cube of events is maintained in the Event Cube version component, which can be seen as an instance of an OLAP server. The data analysis (i.e., the process discovery and analysis) can be performed through the Process Explorer version component, by exploiting the data cube of events.

Component	*Object*	*Type*
Event Log	Process Data	File
Process Cases	Process Data	Data Stream
Event Cube	Data Cube	Database
Process Explorer	Multidimensional Process Models	Result
Process Explorer	Process Patterns	Result

Table 2.4: Multidimensional Process Explorer: characterization of the component's data objects.

[3]For more information about the Apache Solr check: `http://lucene.apache.org/solr/`

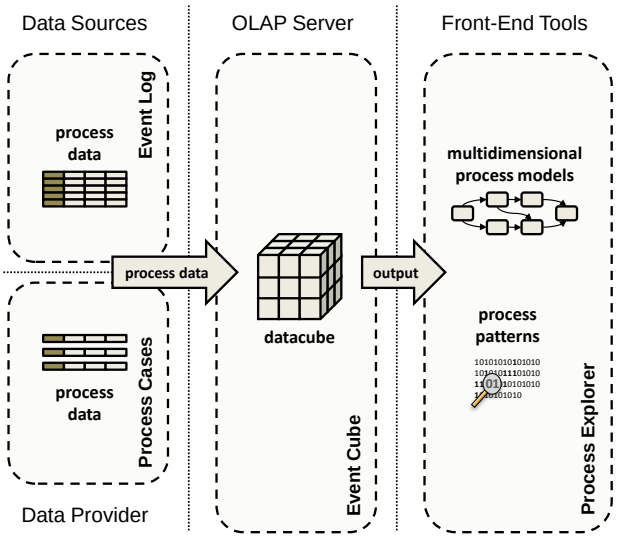


Figure 2.9: Overview of the Multidimensional Process Explorer framework.

In order to present the component diagram of the Multidimensional Process Explorer, we first present in Figure 2.10 an aggregated version of the component diagram of the enterprise search engine. Based on the component diagram of Figure 2.7, this version provides a simplified view of the enterprise search engine by aggregating vendor types components into system type ones. Remark that, since the Multidimensional Process Explorer interacts exclusively with the data provider system, the data provider and the staging area are considered as a single component. The detail level of the descriptions of these component diagrams can also be summarized in Figure 2.5. This time the solid line represents the component diagram of the enterprise search engine, while the dashed line refers to the aggregated version. By combining the component-oriented with the layered architecture styles, we finally present in Figure 2.11 the component diagram of a Multidimensional Process Explorer framework working with an ESAT system. This component diagram is an instantiation of the aggregated version of the component diagram of the enterprise search engine (Figure 2.10). The components represent version components, while the connectors describe how the data flows from one component to another. The component's interfaces are considered implicit in the components.

According to the component diagram in Figure 2.11, there are five version components:

Event Log is a data structure (commonly an XML file) that organizes the process data as sets of traces (process instances), being each trace a set of events. Further details about event logs are provided in Subsection 3.1.1.

Process Data consists of all sorts of data that describe the execution of business processes. Typically, process data are distributed across different operational databases and their exploitation depends on a data provider (for retrieving) and

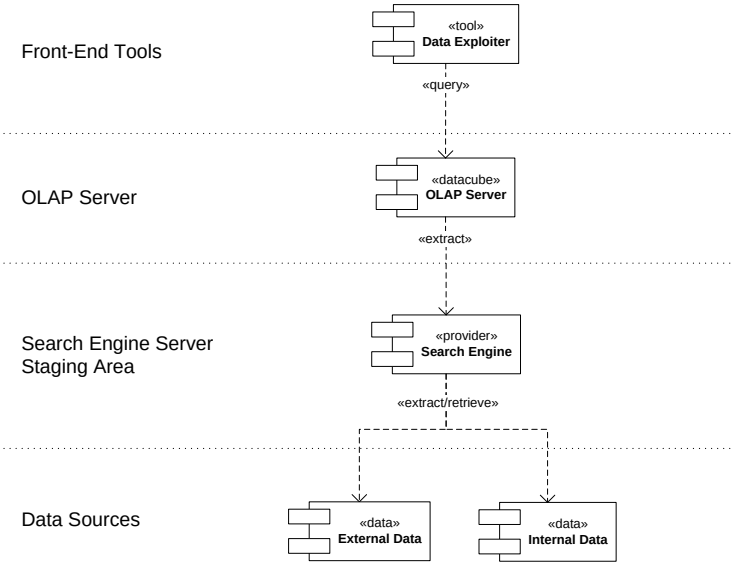


Figure 2.10: Aggregated version of the component diagram of the enterprise search engine.

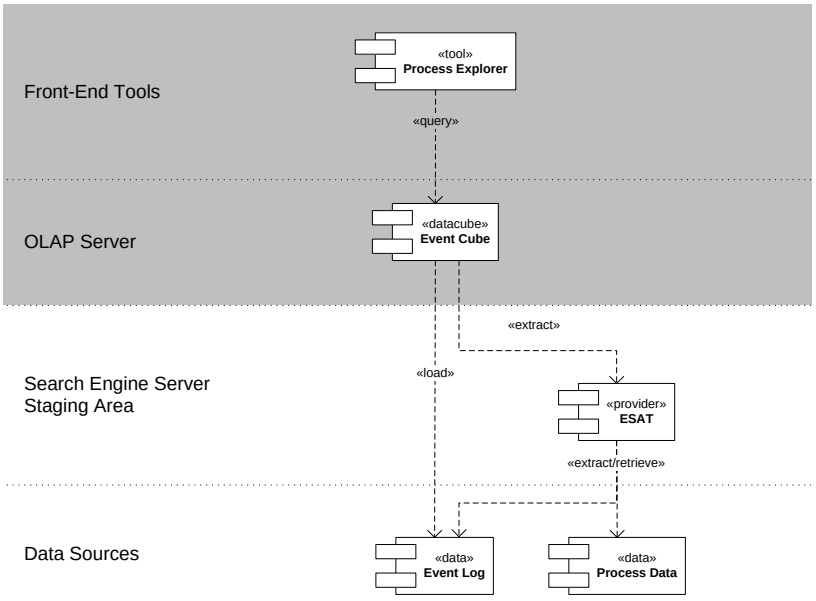


Figure 2.11: Component diagram of the Multidimensional Process Explorer. The gray-colored layers identify core components of the system. The direct connection between the Event Cube and the Event Log components does not reflect the architecture stratification of the Figure 2.10's component diagram but the framework's functionality of working directly on event logs.

transformation services (for normalizing and structuring).

ESAT system is a possible instance of the enterprise search engine architecture. This system may be used to automatically retrieve process data from the data sources.

Event Cube is a data cube of process events that can be used to analyze the business information (process data) under different levels of abstraction.

Process Explorer is an OLAP-based tool for process discovery and analysis. This tool may combine knowledge discovery techniques from data mining, OLAP, and process mining.

Further details about the Multidimensional Process Explorer framework are provided in the next chapters.

2.5 Summary

A concept architecture of a system in which the requirements of the DataFusion project are fulfilled is introduced in this chapter. Two different approaches are identified to establish an effective connection between the data sources and the knowledge discovery techniques: the data warehousing systems and the enterprise search engines. The first one can be seen as the logical option once that this kind of system is optimized to integrate data from heterogeneous sources for decision supporting. However, since NPD processes are dynamic and multidisciplinary, some of the prerequisites for implementing a data warehousing system may not be met. Hence, as an alternative approach, the search engine systems are considered for overcoming these issues. The framework for multidimensional process discovery described in this thesis is introduced as a part of the architecture.

Addressing research question Q', the concept architecture can be used to establish an effective connection between the data sources and the knowledge discovery techniques (e.g., the framework described in this thesis).

Chapter 3

Process Discovery

In this chapter, we present a methodology for discovering business processes. Taking some process data as starting point, it is intended to provide insight into the business process by representing it in such a way that its behavior can be easily understood. To achieve this, a new process representation language is introduced. The level of abstraction provided by this language should not be restricted to the traditional process representation (i.e., process models characterized by activities and their relationships). The dimensionality of the business processes (i.e., the multiple aspects of the business described in the process data) should also be considered in the proposed language. Furthermore, this methodology should be robust enough to work not only on complete and high-quality data from well-defined processes but also on incomplete and noisy data from low-structured processes.

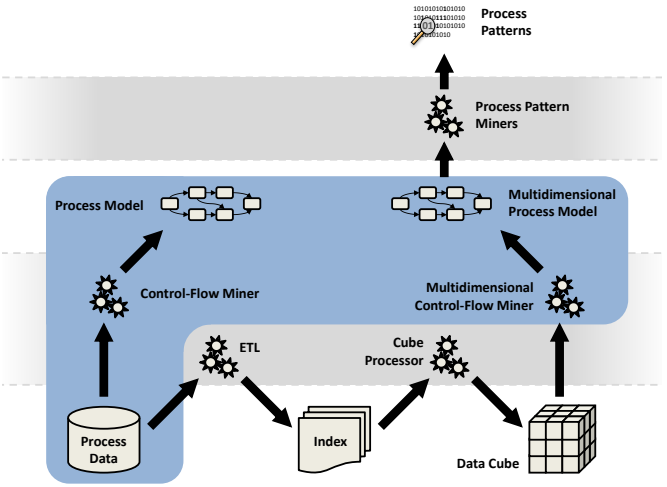


Figure 3.1: Positioning of Chapter 3 with respect to the multidimensional process discovery approach.

Figure 3.1 positions this chapter with respect to the multidimensional process discovery approach. In the next sections, we firstly characterize process data in terms of structure, meaning, and relationships. Then, we define two process representation notations that can be used to present the process data as a process model. The first

notation, the Causal nets, is designed to provide insight into the process by using an easy-to-understand model in which non-trivial control-flow constructs can be represented. The second notation, the multidimensional Causal nets, can be seen as a generalization of the first one once that, instead of simply considering the activities performed in the process (i.e., dimension *Activity*), it can take into account multiple aspects of the process (i.e., other dimensions such as *Resource* or *Time*). Next, we present two process discovery techniques for deriving process models from process data. The first technique, the Flexible Heuristics Miner, is designed to discover a Causal net from processes that can be low structured, eventually dealing with the presence of noise in the process data. The second technique, the Multidimensional Heuristics Miner, is designed to discover a multidimensional Causal net. This technique can be seen as a generalization of the first one once that it can take into account multiple dimensions.

3.1 Process Data

Business process data describe the execution of the different process events of one or more business processes over time. Supported by process aware information systems, process data enclose information about the process cases (instances) and their process events, and, eventually, operational data produced in the business process.

Definition 3.1 (Process Event) *A process event is a fact of the business process (at the most atomic level) that describes the execution of an activity (or task). A process event can be characterized by multiple aspects such as the resource who executed the activity or when the activity was performed. The aspects used to characterize process events are designated as event-based attributes. A process event is uniquely identified by the pair of values* (PID, EID)*, where* PID *is the unique key of the process instance in which the process event was executed, and* EID *is the position of the process event in the process instance.*

Definition 3.2 (Process Instance) *A process instance is a sequence of process events in either an event log or an event stream. Besides the sequence of process events, a process instance can be characterized by multiple aspects such as the product or service processed or the customer for which the process instance was executed. The aspects used to characterize a process instance are designated as instance-based attributes. Every process event inherits the characteristics of the process instance which it is part of. A process instance is uniquely identified by the value* PID*, which is the unique key of the process instance.*

An event log is a data structure where process data are integrated. An event stream may be used to transmit process data from a system to another.

3.1.1 Event Logs

Basically, event logs (or transition logs) organize the process data as sets of process instances. Figure 3.2 presents a generic schema of an event log as a UML class diagram. It is assumed that each process event is always characterized by the attribute *Activity*, which describes the performed activity or task of the process event. Eventually, information about who performed the activity (attribute *Originator*), when the activity was performed (attribute *Timestamp*), and which type the activity is

(attribute *Event Type*) can be provided as typical process event attributes. Generic event-based attributes can be used to characterize other aspects of process events. Similarly, instance-based attributes can be used to describe specific aspects of process instances. Finally, generic attributes can be used to provide details about the event log. For further insight into the structure of event logs see [46, 120, 126].

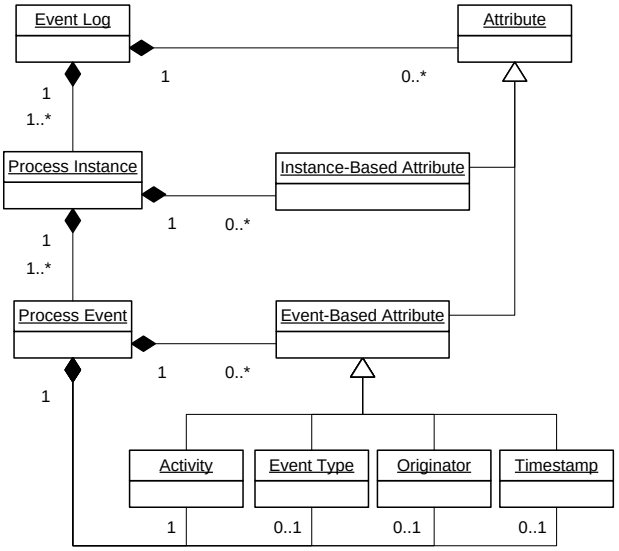


Figure 3.2: Structure of an event log.

Table 3.1 shows an example of an event log. Remark that, in event logs, the process events are grouped by process instance and ordered by execution time.

PID	*Type*	*Activity*	*Originator*	*Timestamp*	*Product*	*Decision*
t_1	•	a	*System*	19-06-2012:14.00	α	
t_1	•	b	*Tester*	19-06-2012:15.10		
t_1	•	e	*Clerk*	19-06-2012:15.45		
t_1	•	d	*Repairer*	20-06-2012:10.30		
t_1	•	g	*Accountant*	20-06-2012:15.35		
t_2	•	a	*System*	20-06-2012:10.05	β	
t_2	•	c	*Analyst*	20-06-2012:11.20		*pos*
t_2	•	f	*Repairer*	20-06-2012:15.30		
...						
t_3	•	a	*System*	20-06-2012:13.45	α	
t_3	•	c	*Analyst*	20-06-2012:16.00		*neg*
...						

Table 3.1: Example of an event log.

The process instance t_1 can be described by the following sequence of process events $\langle e_1 e_2 e_3 e_4 e_5 \rangle$, with e_i being the process event represented in the i^{th} row of Table 3.1.

Eventually, this sequence of events can be projected onto a sequence of attribute values $\langle v_1 v_2 v_3 v_4 v_5 \rangle$, with v_i being the value of a given attribute in the process event e_i. For example, t_1 can be described by the following sequences:

- **Activities**: $\langle abedg \rangle$.
- **Originators**: $\langle (System)(Tester)(Clerk)(Repairer)(Accountant) \rangle$.
- **Products**: $\langle \alpha???? \rangle$, being the wildcard value '?' used to represent the unknown values.
- **Types**: $\langle \bullet \bullet \bullet \bullet \bullet \rangle$.

3.1.2 Event Streams

An event stream is a data stream that consists of a continuous sequence of process events. Unlike in event logs, process events in event streams are not grouped by process instance. Additionally, unless indicated explicitly in the data, a process instance cannot be assumed as complete once that further events of that instance can be received at any point in time.

An operational data store is required to maintain the process data received in the event stream. By grouping the process events by process instance, the stored information can be equated to an event log with the difference that the amount of information may change over time. Like in event logs, the structure of process events depends on the information system that generates the data. Nonetheless, it is assumed that the process data in an event stream is characterized by the same attributes as in event logs (i.e., instance- and event-based attributes). Table 3.2 shows an example of an event stream.

PID	*Type*	*Activity*	*Originator*	*Timestamp*	*Product*	*Decision*
t_1	•	a	*System*	19-06-2012:14.00	α	
t_1	•	b	*Tester*	19-06-2012:15.10		
t_1	•	e	*Clerk*	19-06-2012:15.45		
t_2	•	a	*System*	20-06-2012:10.05	β	
t_1	•	d	*Repairer*	20-06-2012:10.30		
t_2	•	c	*Analyst*	20-06-2012:11.20		*pos*
t_3	•	a	*System*	20-06-2012:13.45	α	
t_2	•	f	*Repairer*	20-06-2012:15.30		
t_1	•	g	*Accountant*	20-06-2012:15.35		
t_3	•	c	*Analyst*	20-06-2012:16.00		*neg*
...						

Table 3.2: Example of an event stream.

Remark that it is assumed that the events in the stream can be unequivocally associated to a process instance. In this example, the association of events to process instances is supported by the attribute PID (the process instance identifier). The process event ordering is determined by the time information (i.e., the timestamps).

3.2 Process Representation

A – business – process model can be seen as an abstraction of how work is done in a specific business. A process model can be defined under the scope of two distinct BPM applications: process modeling and process discovery. In process modeling, the process model is designed by process modelers and used as reference for process management. In process discovery, the process model is discovered from process data that describe the – past – execution of a business process. This discovered process model can be used for assessing the process performance and conformance of running processes.

In this section, we introduce a notation for representing two types of process models discovered from process data. The first type consists of traditional process models in which the business process is represented as groups of process events (grouped by activity) and a set of control-flow constructs. In the process notation, the following constructs are considered to characterize the behavior of the process:

- **Sequence**: identifies – causal – dependencies between process events.
- **Splits**: characterizes how process events activate the execution of other events.
- **Joins**: characterizes how process events are activated by other events to be executed.
- **Loops**: describes repetitions in the execution of one or more process events.
- **Non-Free Choice**: identifies indirect dependencies that constrain the process behavior.

The second type of process model is a generalization of the first one. Instead of grouping process events simply by activity, process events can be grouped by multiple aspects of the business process. Furthermore, constructs can also be characterized by multiple aspects. This second type of process model is designated as multidimensional process model.

3.2.1 Traditional Process Models

A traditional process model (or a one-dimensional process model) is an activity-centric process model that describes the business process in terms of activities and their dependency relations. Transition systems, Petri nets, BPMN, and EPCs are examples of notations for modeling these models. Depending on the notation, process-related information such as resources, time, decisions and rules may also be described in a traditional process model. For an overview of process notations see [93, 110].

Characterized by their simplicity and flexibility, Causal nets (C-nets) are commonly used as process notation in process mining [6, 130].[1] Following the given definition of traditional process model, C-nets consist of a set of activities and, for each activity, a set of possible *input bindings* and a set of possible *output bindings*. Also known as joins and splits, these sets of – input and output – bindings define not only the inputs and outputs of a specific activity but also how these inputs and outputs are activated.

Definition 3.3 (Causal net – strict definition [117]) *A Causal net (C-net) is a tuple* (A, a_i, a_o, D, I, O) *where*

- A *is a finite set of* activities*,*
- $a_i \in A$ *is the* start activity*,*

[1] Causal nets are also known as Heuristics nets.

- $a_o \in A$ *is the* end activity,
- $D \subseteq A \times A$ *is the set of* dependency relations,
- $AS = \{X \in \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$, *is the set of all possible input and output bindings,*[2]
- $I \in A \rightarrow AS$ *defines the set of possible* input bindings *per activity, and*
- $O \in A \rightarrow AS$ *defines the set of possible* output bindings *per activity*

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{as \in I(a_2)} as \wedge a_2 \in \bigcup_{as' \in O(a_1)} as'\}$,
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$,
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$, *and*
- *All activities in the graph* (A, D) *are on a path from* a_i *to* a_o.

The representation of a C-net consists of a graph where nodes represent activities and arcs (or edges) represent dependency relations. Eventually, – non-empty – input and output bindings (joins and splits) can be explicitly represented in the graph (e.g., Figure 3.4). Another representation for splits and joins is a kind of *truth table* where only the possible (positive) cases are described (e.g., Table 3.5). A split (or a join) is represented in the disjunctive normal form (DNF) and may be categorized as one of the following simple types of splits and joins.

- **XOR**: an XOR-split (or an XOR-join) is formed exclusively by single-element bindings (i.e., only one output (or input) is activated at a time). Remark that this definition does not match with the definition of exclusive disjunction in logic.[3]
- **AND**: an AND-split (or an AND-join) is formed exclusively by a single binding in which all elements (inputs or outputs) are activated.
- **OR**: an OR-split (or an OR-join) is formed by all the possible combinations of bindings (i.e., the powerset of elements (inputs or outputs) excluding the empty set).

Figure 3.3 describes the simple types of splits and joins. A split (or a join) that cannot be associated to any of these types is designated as a *complex split* (or a *complex join*). For example, the C-net of Figure 3.4 contains a complex join. The join of activity g expresses that this activity can be activated for execution by

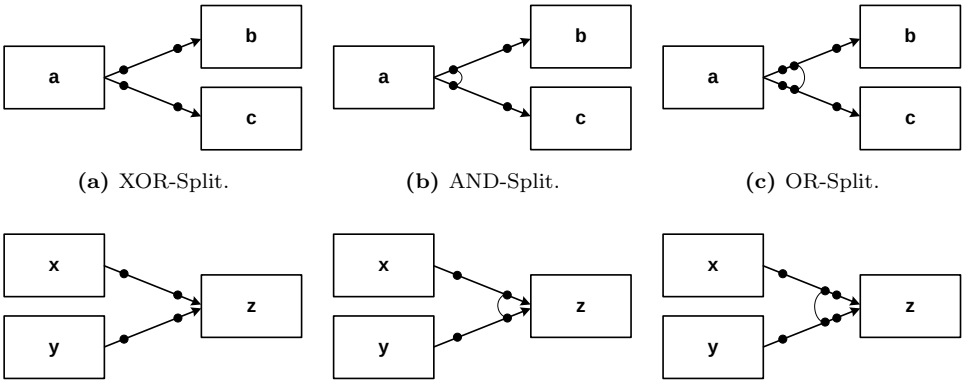
1. only activity e (i.e., $\{e\}$),
2. only activity f (i.e., $\{f\}$),
3. both activities d and e (i.e., $\{d, e\}$), or
4. both activities e and f (i.e., $\{e, f\}$).

Therefore, this join is not an XOR- (the input binding $\{d\}$ is missing), an AND- (the input binding $\{d, e, f\}$ is missing), or an OR-join (the input binding $\{d, f\}$, for instance, is missing).

Figure 3.3a shows an XOR-split in which, after activity a, either activity b or activity c is executed. Figure 3.3b depicts an AND-split in which both activities b and c are executed after activity a. Figure 3.3c illustrates an OR-split in which (i) only

[2] $\mathcal{P}(X) = \{Y | Y \subseteq X\}$ is the powerset of some set X.

[3] In logic, the exclusive disjunction is defined to be true if an odd number of its elements are true, and false otherwise.



(a) XOR-Split. **(b)** AND-Split. **(c)** OR-Split.

(d) XOR-Join. **(e)** AND-Join. **(f)** OR-Join.

Figure 3.3: The different types of splits and joins.

activity b, (ii) only activity c, or (iii) both activities b and c are executed after activity a. Analogously, figures 3.3d, 3.3e, and 3.3f present examples of XOR-, AND-, and OR-joins with regard to activity z. Remark that these types of splits and joins can be combined in order to describe more complex process behavior.

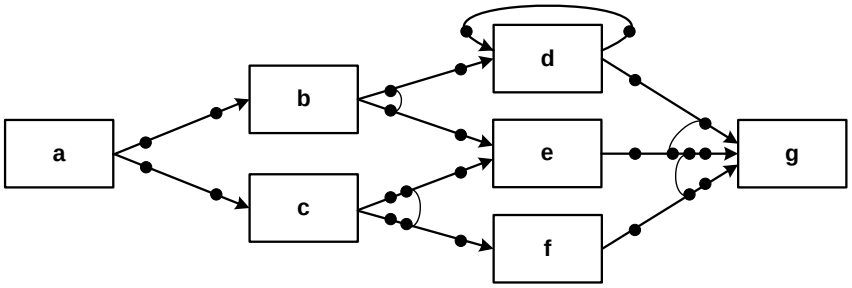


Figure 3.4: Example of a C-net.

The C-net of Figure 3.4 can be described as follows. The set of activities is $A = \{a, b, c, d, e, f, g\}$. These activities are represented by the nodes of the process model. The dependency relations depicted by the edges of the process model are given by $D = \{(a, b), (a, c), (b, d), (b, e), (c, e), (c, f), (d, d), (d, g), (e, g), (f, g)\}$. Functions $I(a)$ and $O(a)$ identify the sets of possible inputs and output bindings of activity a. Table 3.3 shows all inputs and output bindings in the process model of Figure 3.4. Activity a is the – unique – start activity (i.e., $a_i = a$). This means that activity A has no input bindings (i.e., $I(a) = \{\emptyset\}$) and, thus, cannot be executed more than once in a process instance. Analogously, activity g is the – unique – end activity (i.e., $a_o = g$). This activity also cannot be executed more than once in a process instance because its set of possible output bindings is an empty set of activities (i.e., $O(g) = \{\emptyset\}$).

C-nets can describe explicitly how activities are related to each other by defining dependency relations. This is represented in the process model by connecting nodes (activities) with edges (dependency relations). However, although the input or output

Input Bindings	*Activity*	*Output Bindings*
$\{\emptyset\}$	a	$\{\{b\},\{c\}\}$
$\{\{a\}\}$	b	$\{\{d,e\}\}$
$\{\{a\}\}$	c	$\{\{e\},\{f\},\{e,f\}\}$
$\{\{b\},\{d\}\}$	d	$\{\{d\},\{g\}\}$
$\{\{b\},\{c\}\}$	e	$\{\{g\}\}$
$\{\{c\}\}$	f	$\{\{g\}\}$
$\{\{e\},\{f\},\{d,e\},\{e,f\}\}$	g	$\{\emptyset\}$

Table 3.3: All input and output bindings of the C-net of Figure 3.4.

relations of an activity can be characterized as input or output bindings, it is not possible to describe which input binding activated a specific output binding. In order to overcome this issue, the *activity binding* concept is defined to characterize a relationship between input and output bindings.

Definition 3.4 (Activity Binding) *Let $M = (A, a_i, a_o, D, I, O)$ be a C-net according to Definition 3.3. An activity binding B^x of an activity $x \in A$ is defined by the tuple $(as^I, as^O)^x$ where $as^I \in I(x)$ and $as^O \in O(x)$ are the input and output bindings. Both input and output bindings are powersets of A.*

Table 3.4 describes all activity bindings in the process model of Figure 3.4. Note that a process instance can be translated into a sequence of activity bindings. For example, the process instance $\langle abdedg \rangle$ can be described as the sequence of activity bindings $\langle(\emptyset,\{b\})^a\ (\{a\},\{d,e\})^b\ (\{b\},\{d\})^d\ (\{b\},\{g\})^e\ (\{d\},\{g\})^d\ (\{d,e\},\emptyset)^g\rangle$.

Activity	*Activity Bindings*
a	$(\emptyset,\{b\})^a$ and $(\emptyset,\{c\})^a$
b	$(\{a\},\{d,e\})^b$
c	$(\{a\},\{e\})^c$, $(\{a\},\{f\})^c$ and $(\{a\},\{e,f\})^c$
d	$(\{b\},\{g\})^d$, $(\{b\},\{d\})^d$, $(\{d\},\{d\})^d$ and $(\{d\},\{g\})^d$
e	$(\{b\},\{g\})^e$ and $(\{c\},\{g\})^e$
f	$(\{c\},\{g\})^f$
g	$(\{d,e\},\emptyset)^g$, $(\{e\},\emptyset)^g$, $(\{f\},\emptyset)^g$ and $(\{e,f\},\emptyset)^g$

Table 3.4: All activity bindings of the C-net of Figure 3.4.

The strict definition of C-net assumes the existence of unique and exclusive start and end activities. This means that the C-net cannot hold more than one start and end activities, and these cannot be executed more than once in the same process instance. Furthermore, all activities in the C-net must be reached by the start activity, and reach the end activity. These assumptions are valid if the business process is well-defined and there are no isolated sub-processes. One effective solution for overcoming this issue is to add artificial start and end activities into every process instance. However, these extra activities will increase the number of objects in the process model, turning the process analysis more challenging. This issue is even more evident in process models from low-structured processes. Therefore, in order to keep the resulting process model as simple as possible, a *relaxed definition* of C-net is introduced. The relaxed definition consists of a generalization of the strict one with regard to the start and end activities.

Multiple start and end activities are allowed in this type of C-net. This means that an activity may be characterized by both empty and non-empty input and output bindings.

Definition 3.5 (Causal net – relaxed definition) *A Causal net (C-net) is a tuple* (A, A_i, A_o, D, I, O) *where*

- A *is a finite set of* activities,
- $A_i \subseteq A$ *is the set of* start activities,
- $A_o \subseteq A$ *is the set of* end activities,
- $D \subseteq A \times A$ *is the set of* dependency relations,
- $I \in A \to \mathcal{P}(\mathcal{P}(A))$ *defines the set of possible* input bindings *per activity, and*
- $O \in A \to \mathcal{P}(\mathcal{P}(A))$ *defines the set of possible* output bindings *per activity*

such that

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup_{x \in I(a_2)} x \wedge a_2 \in \bigcup_{y \in O(a_1)} y\}$,
- $A_i = \{a \in A \mid I(a) \supseteq \{\emptyset\}\}$,
- $A_o = \{a \in A \mid O(a) \supseteq \{\emptyset\}\}$, *and*
- *All activities in the graph* (A, D) *are on a path from a start activity* $a_i \in A_i$ *to an end activity* $a_o \in A_o$.

Remark that a C-net defined according to the relaxed definition can easily be converted into a C-net conforming the strict definition (and the other way around).

One of the limitations of C-nets is the lack of measuring information about the elements described in the process model. For example, all activities in the process model are presented as equi-relevant (i.e., the model does not provide explicit information to assess the relevance of activities). Therefore, low frequent cases (possibly originated by noise) are thus impossible to identify. By using the process data from which the process model is discovered, it is possible to easily compute frequency information of activities, dependency relations, and input and output bindings. By annotating the process model with this information, it is possible to provide insight into the relevance of the elements described in the model. Consequently, cases of noise can be directly identified. A C-net in which the frequency of its elements can be computed and described in the process model is designated as an *augmented Causal net* (aC-net).

Definition 3.6 (Augmented Causal net) *An augmented Causal net (aC-net) is a tuple* (A, A_i, A_o, D, I, O) *according to the relaxed definition of C-net (cf. Definition 3.5) for which*

- $freq : A \to \mathbb{N}$ *is the activity frequency function,*
- $freq : D \to \mathbb{N}$ *is the dependency relation frequency function,*
- $freq : I \to \mathbb{N}$ *is the input binding frequency function, and*
- $freq : O \to \mathbb{N}$ *is the output binding frequency function.*

Figure 3.5 presents the corresponding aC-net of the C-net of Figure 3.4. In this annotated process model, the numerical information in the nodes (in *Italics*) represent the activity frequency (i.e., the number of times the activity appeared in the process data). Similarly, the numerical information in the edges represent the number of times a transition from an activity to another occurred. Information about input and output bindings is not explicitly presented in this process model.

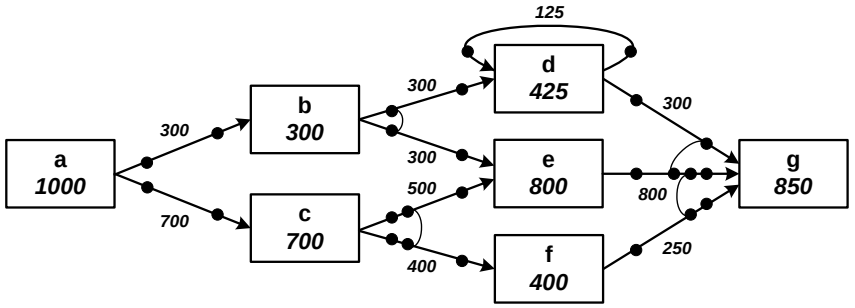


Figure 3.5: Example of an aC-net with one start activity (a) and two end activities (f and g).

The aC-net of Figure 3.5 can be described as follows. $A = \{a, b, c, d, e, f, g\}$ and $D = \{(a, b), (a, c), (b, d), (b, e), (c, e), (c, f), (d, d), (d, g), (e, g), (f, g)\}$ are the sets of activities and dependency relations. Like in C-nets, edges depict dependency relations and nodes represent activities. Also, functions $I(a)$ and $O(a)$ identify the sets of possible inputs and output bindings of activity a. Provided in the definition of aC-net, function *freq* determines the frequency of activities, dependency relations, and input and output bindings. In the process model, the frequency information of activities and dependency relations is provided in nodes and edges. The frequency information of input and output bindings is not implicitly represented in the model but in splits and joins tables. Table 3.5 describes the splits and joins tables of activity d. Each entry of these tables identify the binding's activated inputs or outputs as well as the binding's frequency information. Remark that splits and joins tables can be represented as multisets of input and output bindings.[4] Table 3.6 shows all inputs and output bindings and corresponding frequencies in the process model of Figure 3.5. Activity a is the only start activity (i.e., $A_i = \{a\}$), while activities f and g are the end activities (i.e., $A_o = \{f, g\}$). Unlike g, activity f is not an exclusive end activity once that its set of possible output bindings contains a non-empty set of activities (i.e., $\{g\} \in O(f)$).

Binding	*Inputs*		*Frequency*
	b	d	
$\{b\}$	✓		300
$\{d\}$		✓	125

(a) Joins.

Binding	*Outputs*		*Frequency*
	d	g	
$\{g\}$		✓	300
$\{d\}$	✓		125

(b) Splits.

Table 3.5: Input and output bindings of activity d.

3.2.2 Multidimensional Process Models

A multidimensional process model is a generic process model that describes the business process in terms of groups of process events and their dependency relations. Groups of process events as well as dependency relations can be constrained by multiple di-

[4] A multiset (or bag) is a generalization of a set in which each element may occur multiple times. For example, $[a^1, b^2, c^3]$ is a multiset with six elements (one a, two b's, and three c's).

Input Bindings	*Activity*	*Output Bindings*
$[\emptyset^{1000}]$	a	$[\{b\}^{300}, \{c\}^{700}]$
$[\{a\}^{300}]$	b	$[\{d, e\}^{300}]$
$[\{a\}^{700}]$	c	$[\{e\}^{300}, \{f\}^{200}, \{e, f\}^{200}]$
$[\{b\}^{300}, \{d\}^{125}]$	d	$[\{d\}^{125}, \{g\}^{300}]$
$[\{b\}^{300}, \{c\}^{500}]$	e	$[\{g\}^{800}]$
$[\{c\}^{400}]$	f	$[\{g\}^{250}, \emptyset^{150}]$
$[\{e\}^{300}, \{f\}^{50}, \{d, e\}^{300}, \{e, f\}^{200}]$	g	$[\emptyset^{850}]$

Table 3.6: All input and output bindings – and corresponding frequencies – of the aC-net of Figure 3.5.

mensions. This means that multidimensional process models are a generalization of traditional process models. By grouping process events by activity and using unconstrained dependency relations, it is possible to represent in a multidimensional process model the same information as the one that can be represented in a traditional process model.

A dimension is a particular aspect of the business process, which (i) is represented in the process data through a data element (attribute), or (ii) can be derived from the process data through some data transformation function. The different values used to characterize the dimension are designated as dimension values. Typically categorical, dimension values can be characterized as:

- **Simple values**, which consist of a single value representing a concrete fact of the business process.
- **Transition values**, which consist of two simple values describing a transition from a fact of the business process to another (i.e., a dependency relation).

Definition 3.7 (Dimension and Dimension Value) *Let $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ be the set of dimensions. If $D_i \in \mathcal{D}$ then*

- *$\{v_{i1}, v_{i2}, ..., v_{im}\}$ are the **simple values** of D_i, and*
- *$\{v_{i1}, v_{i2}, ..., v_{im}\} \times \{v_{i1}, v_{i2}, ..., v_{im}\}$ are the **transition values** of D_i.*

Since the same values can appear in different dimensions, dimension values are denoted as:

- ***Simple values**: D_i:v_{ik} ($i \in \{1, ..., n\}$ and $k \in \{1, ..., m\}$).*
- ***Transition values**: D_i is D_i:$v_{ik} \rightarrow v_{ij}$ ($i \in \{1, ..., n\}$ and $k, j \in \{1, ..., m\}$). If $k = j$ then the transition value can be considered as a simple value.*

If $D_i \in \mathcal{D}$ then the set of simple values of D_i is determined by the function values(D_i). *The cardinality of D_i (i.e., the number of simple values in D_i) is determined by the function* card(D_i).

As an example of a simple value, the originator of the first process event in Table 3.1 (first row) can be identified as $Originator$:$System$. As an example of a transition value, the transition of originators from the first to the second process events in the same table (first and second rows) can be described as $Originator$:$System \rightarrow Tester$.

Considering the C-net notation, the representation of a multidimensional process model consists of a multigraph (i.e., a graph in which nodes can be connected by parallel edges) where nodes represent groups of process events and arcs (or edges)

represent constrained dependency relations. Nodes are identified by a set of simple values that describe the characteristics of the represented process events. Edges are constrained by a set of transition values that describe the characteristics of the existing transitions. This means that more than one transition from a node to another can exist. Figure 3.6 shows three multidimensional models representing the same subprocess through different perspectives. In terms of dimensions, these submodels can be described as follows. $\mathcal{D} = \{Task, Resource, Product\}$ is the set of dimensions in all three submodels. Since $\mathcal{D}$ contains three dimensions, these submodels can be considered as three-dimensional submodels. In terms of dimension values, each dimension contains two distinct values (i.e., $values(Task) = \{T_1, T_2\}$, $values(Resource) = \{R_1, R_2\}$, and $values(Product) = \{P_1, P_2\}$).

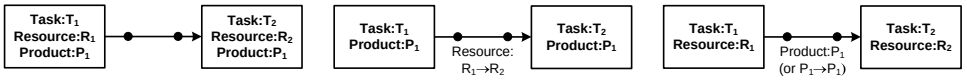


Figure 3.6: Three multidimensional submodels representing different perspectives of the same subprocess.

Each node of the submodel of Figure 3.6a is characterized by three dimension values (simple values). $Task$:T_1, $Resource$:R_1, and $Product$:P_1 are the values in the source node. This submodel is not described by any transition value once that its transition (edge) is not characterized by any dimension. The same does not happen in the other submodels. In the submodel of Figure 3.6b, the transition value $Resource$:$R_1 \rightarrow R_2$ is used to describe the transition in terms of resources. In the submodel of Figure 3.6c, the transition value $Product$:$P_1 \rightarrow P_1$ is used to describe the transition in terms of products. Once that this transition value does not describe an actual transition, $Product$:$P_1 \rightarrow P_1$ may be replaced by the simple value $Product$:P_1.

In a multidimensional process model, each dimension value acts as a filtering condition of process events or transitions within the process. For example, the source node of Figure 3.6a represents the group of process events in which the task T_1 on product P_1 was performed by resource R_1. The set of dimension values used to identify a group of process events is designated as *event constraint*. The edge of Figure 3.6b describes the transition of resources from the group of process events in which task T_1 was performed on product P_1 to the one in which task T_2 was performed on the same product. The set of dimension values used to describe a transition is designated as *workflow constraint*.

Definition 3.8 (Constraint) *Let $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ be the set of dimensions, $\mathcal{V}^S$ the set of all simple values, and $\mathcal{V}^T$ the set of all transition values. $\mathcal{V}^S$ and $\mathcal{V}^T$ are defined as follows.*

$$\mathcal{V}^S = \{\ D_i : v_{ik} \mid D_i \in \mathcal{D} \wedge i \in \{1, ..., n\} \wedge k \in \{1, ..., m\}\ \}$$
$$\mathcal{V}^T = \{\ D_i : v_{ik} \rightarrow v_{ij} \mid D_i \in \mathcal{D} \wedge i \in \{1, ..., n\} \wedge k, j \in \{1, ..., m\}\ \}$$

A constraint is a set of dimension values from distinct dimensions (i.e., a dimension cannot be represented in a constraint by more than one dimension value) that can be used to filter (or to partition) the process data. There are two different types of constraints:

- ***Event constraint**: Let $\mathcal{C}^S$ be the set of all constraints based on simple values that can defined as*

$$\mathcal{C}^S = \{\ X \mid X \subseteq \mathcal{V}^S \wedge \forall_{i,j,k,l}[D_i : v_{ik} \in X \wedge D_j : v_{jl} \in X \wedge i = j \rightarrow v_{ik} = v_{jl}]\ \}.$$

An event constraint $C^E \in \mathcal{C}^S$ is a set of simple values can be used to describe one or more process events or facts of the business process.

- ***Workflow constraint**: Let $\mathcal{C}^T$ be the set of all constraints based on transition values that can be defined as*

$$\mathcal{C}^T = \{\ X \mid X \subseteq \mathcal{V}^T \wedge \forall_{i,j,k,k',l,l'}[D_i : v_{ik} \rightarrow v_{ik'} \in X \wedge D_j : v_{jl} \rightarrow v_{jl'} \in X \wedge \\ \wedge\ i = j \rightarrow v_{ik} = v_{jl} \wedge v_{ik'} = v_{jl'}]\ \}.$$

A workflow constraint $C^W \in \mathcal{C}^T$ is a set of transition values that can be used to describe a specific transition between process events or facts of the business process.

If a constraint is an empty set (i.e., $C^E = \emptyset$ or $C^W = \emptyset$) then no filtering action is applied to the process data. Otherwise, a partition of the process data is generated containing all entries in which all dimension values of the constraint are satisfied. Remark that a constraint holds zero or one dimension values from each dimension.

Table 3.7 summarizes the different constraints in Figure 3.6. Representing groups of process events, nodes are characterized by event constraints. Representing transitions, edges are characterized by workflow constraints.

	Source Node	*Target Node*
(a)	$\{Task{:}T_1, Resource{:}R_1, Product{:}P_1\}$	$\{Task{:}T_2, Resource{:}R_2, Product{:}P_1\}$
(b)	$\{Task{:}T_1, Product{:}P_1\}$	$\{Task{:}T_2, Product{:}P_1\}$
(c)	$\{Task{:}T_1, Resource{:}R_1\}$	$\{Task{:}T_2, Resource{:}R_2\}$

(a) Event constraints.

	Edge
(a)	$\emptyset$
(b)	$\{Resource{:}R_1 \rightarrow R_2\}$
(c)	$\{Product{:}P_1\}$

(b) Workflow constraints.

Table 3.7: The event and workflow constraints in Figure 3.6.

A process model describes groups of process events, which are unique facts of the business process described in the process data. For example, a traditional process model describes groups of process events characterized by the same activity (i.e., one of the process event's attributes). In these one-dimensional models, the term *activity* is used to identify the groups of process events. In multidimensional process models, a group of process events can be characterized by multiple aspects. The concept *event occurrence* is defined to describe a group of process events that satisfy specific characteristics (event constraints).

Definition 3.9 (Event Occurrence) *In a multidimensional process model, an event occurrence E is a group of process events characterized by an event constraint (cf.*

Definition 3.8). Analogously, a transition from an event occurrence E_1 to another E_2 is defined by a workflow constraint. Unlike traditional process models, there may exist more than one transition between E_1 and E_2.

Every node of the submodels of Figure 3.6 represents an event occurrence. For example, in Figure 3.6a, the event occurrence represented by the source node describes the group of process events in which task T_1 processed the product P_1, and was performed by the resource R_1. The same logic applies to the other event occurrence (target node). The transition represented by the edge is not characterized by any dimension (i.e., it is not constrained). In Figure 3.6b, the edge describes a transition of resources from an event occurrence to another. This means that resource R_1 performed the first event occurrence, while R_2 performed the other one. In Figure 3.6c, the transition describes that the product P_1 was processed in both event occurrences of the submodel (i.e., there is no transition of product).

Considering the concepts introduced above, a multidimensional version of Causal nets can be finally defined as a notation for modeling multidimensional process models. In multidimensional Causal nets,

- event occurrences are considered instead of activities,
- input and output bindings – and, consequently, dependency relations – can be characterized by workflow constraints, and
- measuring information of event occurrences, dependency relations, and bindings is provided according to a specific summarization function.

Note that, as a generalization of aC-nets, multidimensional Causal nets preserve the semantics of the traditional Causal nets.

Definition 3.10 (Multidimensional Causal net) *A multidimensional Causal net (mC-net) is a tuple (E, C, E_i, E_o, D, B) where*

- *E is a finite set of* event occurrences,
- *$E_i \subseteq E$ is the set of* start event occurrences,
- *$E_o \subseteq E$ is the set of* end event occurrences,
- *C is a finite set of* workflow constraints,
- *$D \subseteq E \times E \times C$ is the set of* dependency relations,
- *$I \in E \to \mathcal{P}(\mathcal{P}(E \times C))$ defines the set of possible* input bindings *per event occurrence,*
- *$O \in E \to \mathcal{P}(\mathcal{P}(E \times C))$ defines the set of possible* output bindings *per event occurrence, and*
- *$B \in E \to I \times O$ defines the set of possible* event occurrence bindings *per event occurrence*

such that

- *$D = \{(e_1, e_2, c) \in E \times E \times C \mid (e_1, c) \in \bigcup_{x \in I(e_2)} x \wedge (e_2, c) \in \bigcup_{y \in O(e_1)} y\}$,*
- *$E_i = \{e \in E \mid I(e) \supseteq \{\emptyset\}\}$,*
- *$E_o = \{e \in E \mid O(e) \supseteq \{\emptyset\}\}$, and*
- *All activities in the graph (E, D) are on a path from a start event occurrence $e_i \in E_i$ to an end event occurrence $e_o \in E_o$.*

Any event occurrence, dependency relation, or binding of a mC-net can be characterized by

- $f^{sum} : E \rightarrow \mathbb{R}$*, which is the event occurrence summarization function,*
- $f^{sum} : D \rightarrow \mathbb{R}$*, which is the dependency relation summarization function,*
- $f^{sum} : I \rightarrow \mathbb{R}$*, which is the input binding summarization function,*
- $f^{sum} : O \rightarrow \mathbb{R}$*, which is the output binding summarization function, and*
- $f^{sum} : B \rightarrow \mathbb{R}$*, which is the event occurrence binding summarization function.*

Figure 3.7 presents the corresponding mC-net of the aC-net of Figure 3.5. In this multidimensional process model, the numerical information in the nodes (in *Italics*) represent the activity frequency (i.e., the number of times the activity appeared in the process data). Information about input and output bindings is not explicitly presented in this process model. Edges represented in different colors are characterized by different workflow constraints. In this case, workflow constraints represent the dimension $Type$ (e.g., the type of product or service being processed). Note that in this process there are distinct process behaviors for different types. For example, the type represented by the gray color is never processed in activities b and d, while the one represented by the black color can be processed in any activity.

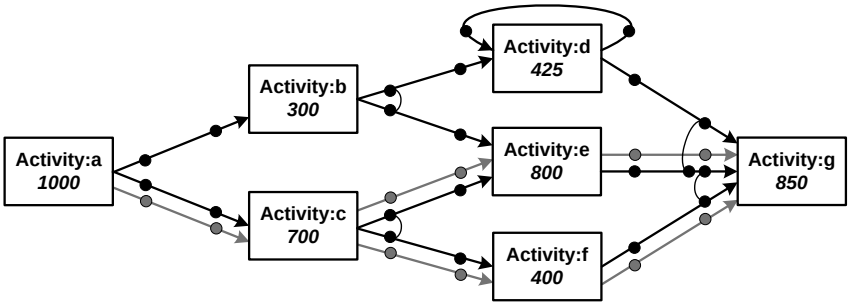


Figure 3.7: A mC-net with 1-D nodes (dimension *Activity*) and 1-D edges (dimension *Type*).

The mC-net of Figure 3.7 can be described as follows. $\mathcal{D} = \{Activity, Type\}$ is the set of dimensions in the process model. Since $\mathcal{D}$ contains two dimensions, this mC-net can be considered as a two-dimensional process model. Dimension *Activity* and its values ($values(Activity) = \{a, b, c, d, e, f, g\}$) are used as event constraints, while dimension $Type$ and its values ($values(Type) = \{I, II\}$) are used as workflow constraints. The set of event occurrences is $E = \{\{Activity{:}a\}, \{Activity{:}b\}, ..., \{Activity{:}g\}\}$, being each occurrence defined by an event constraint, and summarized by the event occurrence frequency function (i.e., $f^{sum}(e) = freq(e)$, with $e \in E$). The set of dependency relations is

$$D = \{ (\{Activity{:}a\}, \{Activity{:}b\}, \{Type{:}I\}), (\{Activity{:}a\}, \{Activity{:}c\}, \{Type{:}II\}), ..., (\{Activity{:}f\}, \{Activity{:}g\}, \{Type{:}I\}), (\{Activity{:}f\}, \{Activity{:}g\}, \{Type{:}II\}) \},$$

being each relation constrained by one of the values of dimension $Type$ (i.e., either I or II). Functions $I(\{Activity{:}a\})$, $O(\{Activity{:}a\})$, and $B(\{Activity{:}a\})$ identify the sets of possible inputs, output, and event occurrence bindings of event occurrence

{*Activity*:*a*}. Table 3.8 shows all inputs and output bindings (and corresponding frequencies) in the process model of Figure 3.7. The possible event occurrence bindings are described in Table 3.9. Event occurrence {*Activity*:*a*} is the only start event occurrence (i.e., $E_i = \{\{Activity{:}a\}\}$), while event occurrences {*Activity*:*f*} and {*Activity*:*g*} are the end event occurrences (i.e., $E_o = \{\{Activity{:}f\}, \{Activity{:}g\}\}$).

Depending on the assignment of dimensions to nodes and edges, the same n-dimensional process model can be presented in 2^n different perspectives. Presenting the same information in different process views, these perspectives can be used to emphasize the different aspects of the business toward process events (nodes) or process behavior (edges). For example, Figure 3.7 presents one out of four possible perspectives of the two-dimensional process model over the dimensions *Activity* and *Type*. In this perspective, dimension *Activity* is emphasized toward process events, while dimension *Type* toward process behavior. Figure 3.8 presents an opposite perspective to Figure 3.7 where the dimension emphases are swapped (i.e., dimension *Type* is emphasized toward process events, while dimension *Activity* toward process behavior). The other possible perspectives of the same multidimensional process model are presented in figures 3.9 and 3.10. In Figure 3.9, both dimensions *Activity* and *type* are emphasized toward process behavior. In Figure 3.10, both dimensions *Activity* and *type* are emphasized toward process events. Remark that all of these perspectives are only given as an illustration of the concept.

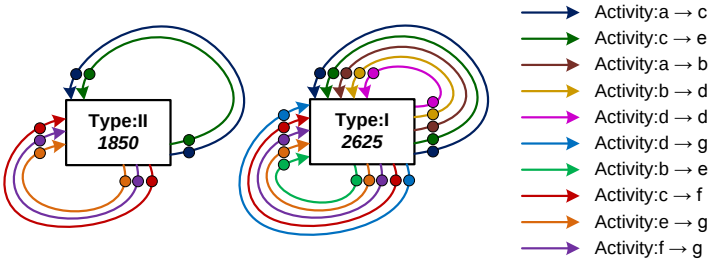


Figure 3.8: A mC-net with 1-D nodes (dimension *Type*) and 1-D edges (dimension *Activity*).

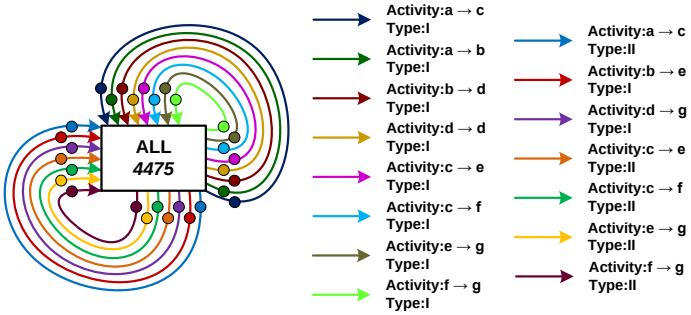


Figure 3.9: A mC-net with 0-D nodes and 2-D edges (dimensions *Activity* and *Type*).

Event Occurrence	*Input Bindings*
$\{Activity{:}a\}$	$[\ \emptyset^{1000}\ ]$
$\{Activity{:}b\}$	$[\ \{\ (\{Activity{:}a\}, \{Type{:}I\})\ \}^{300}\ ]$
$\{Activity{:}c\}$	$[\ \{\ (\{Activity{:}a\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}a\}, \{Type{:}II\})\ \}^{500}\ ]$
$\{Activity{:}d\}$	$[\ \{\ (\{Activity{:}b\}, \{Type{:}I\})\ \}^{300}\ ,\ \{\ (\{Activity{:}d\}, \{Type{:}I\})\ \}^{125}\ ]$
$\{Activity{:}e\}$	$[\ \{\ (\{Activity{:}b\}, \{Type{:}I\})\ \}^{300}\ ,\ \{\ (\{Activity{:}c\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}c\}, \{Type{:}II\})\ \}^{300}\ ]$
$\{Activity{:}f\}$	$[\ \{\ (\{Activity{:}c\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}c\}, \{Type{:}II\})\ \}^{200}\ ]$
$\{Activity{:}g\}$	$[\ \{\ (\{Activity{:}e\}, \{Type{:}II\})\ \}^{300}\ ,\ \{\ (\{Activity{:}d\}, \{Type{:}I\}), (\{Activity{:}e\}, \{Type{:}I\})\ \}^{300}\ ,$ $\{\ (\{Activity{:}e\}, \{Type{:}I\}), (\{Activity{:}f\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}f\}, \{Type{:}II\})\ \}^{50}\ ]$

(a) Input bindings.

Event Occurrence	*Output Bindings*
$\{Activity{:}a\}$	$[\ \{\ (\{Activity{:}b\}, \{Type{:}I\})\ \}^{300}\ ,\ \{\ (\{Activity{:}c\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}c\}, \{Type{:}II\})\ \}^{500}\ ]$
$\{Activity{:}b\}$	$[\ \{\ (\{Activity{:}d\}, \{Type{:}I\}), (\{Activity{:}e\}, \{Type{:}I\})\ \}^{300}\ ]$
$\{Activity{:}c\}$	$[\ \{\ (\{Activity{:}e\}, \{Type{:}II\})\ \}^{300}\ ,\ \{\ (\{Activity{:}f\}, \{Type{:}II\})\ \}^{200}\ ,$ $\{\ (\{Activity{:}e\}, \{Type{:}I\}), (\{Activity{:}f\}, \{Type{:}I\})\ \}^{200}\ ]$
$\{Activity{:}d\}$	$[\ \{\ (\{Activity{:}d\}, \{Type{:}I\})\ \}^{125}\ ,\ \{\ (\{Activity{:}g\}, \{Type{:}I\})\ \}^{300}\ ]$
$\{Activity{:}e\}$	$[\ \{\ (\{Activity{:}g\}, \{Type{:}I\})\ \}^{500}\ ,\ \{\ (\{Activity{:}g\}, \{Type{:}II\})\ \}^{300}\ ]$
$\{Activity{:}f\}$	$[\ \{\ (\{Activity{:}g\}, \{Type{:}I\})\ \}^{200}\ ,\ \{\ (\{Activity{:}g\}, \{Type{:}II\})\ \}^{50}\ ,\ \emptyset^{150}\ ]$
$\{Activity{:}g\}$	$[\ \emptyset^{850}\ ]$

(b) Output bindings.

Table 3.8: All inputs and output bindings – and corresponding frequencies – of the mC-net of Figure 3.7.

Event Occurrence	*Event Occurrence Bindings*
$\{Activity{:}a\}$	$[\ (\ \emptyset\ ,\ \{(\ \{Activity{:}b\},\{Type{:}I\}\ )\}\ )^{300},\ (\ \emptyset\ ,\ \{(\ \{Activity{:}c\},\{Type{:}I\}\ )\}\ )^{200},\ (\ \emptyset\ ,\ \{(\ \{Activity{:}c\},\{Type{:}II\}\ )\}\ )^{500}\ ]$
$\{Activity{:}b\}$	$[\ (\ \{(\ \{Activity{:}a\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ ),(\ \{Activity{:}e\},\{Type{:}I\}\ )\}\ )^{300}\ ]$
$\{Activity{:}c\}$	$[\ (\ \{(\ \{Activity{:}a\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}e\},\{Type{:}I\}\ ),(\ \{Activity{:}f\},\{Type{:}I\}\ )\}\ )^{200},\ (\ \{(\ \{Activity{:}a\},\{Type{:}II\}\ )\}\ ,\ \{(\ \{Activity{:}e\},\{Type{:}II\}\ )\}\ )^{300},\ (\ \{(\ \{Activity{:}a\},\{Type{:}II\}\ )\}\ ,\ \{(\ \{Activity{:}f\},\{Type{:}II\}\ )\}\ )^{200}\ ]$
$\{Activity{:}d\}$	$[\ (\ \{(\ \{Activity{:}b\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}I\}\ )\}\ )^{200},\ (\ \{(\ \{Activity{:}b\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ )\}\ )^{100},\ (\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ )\}\ )^{25},\ (\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}I\}\ )\}\ )^{100}\ ]$
$\{Activity{:}e\}$	$[\ (\ \{Activity{:}b\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}I\}\ )\}\ )^{300},\ (\ \{(\ \{Activity{:}c\},\{Type{:}II\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}II\}\ )\}\ )^{300},\ (\ \{(\ \{Activity{:}c\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}I\}\ )\}\ )^{200}\ ]$
$\{Activity{:}f\}$	$[\ (\ \{(\ \{Activity{:}c\},\{Type{:}I\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}I\}\ )\}\ )^{200},\ (\ \{(\ \{Activity{:}c\},\{Type{:}II\}\ )\}\ ,\ \{(\ \{Activity{:}g\},\{Type{:}II\}\ )\}\ )^{50},\ (\ \{(\ \{Activity{:}c\},\{Type{:}II\}\ )\}\ ,\ \emptyset\ )^{150}\ ]$
$\{Activity{:}g\}$	$[\ (\ \{(\ \{Activity{:}d\},\{Type{:}I\}\ ),(\ \{Activity{:}e\},\{Type{:}I\}\ )\}\ ,\ \emptyset\ )^{300},\ (\ \{(\ \{Activity{:}e\},\{Type{:}II\}\ )\}\ ,\ \emptyset\ )^{300},\ (\ \{(\ \{Activity{:}e\},\{Type{:}I\}\ ),(\ \{Activity{:}f\},\{Type{:}I\}\ )\}\ ,\ \emptyset\ )^{200},\ (\ \{(\ \{Activity{:}f\},\{Type{:}II\}\ )\}\ ,\ \emptyset\ )^{50}\ ]$

Table 3.9: All event occurrence bindings – and corresponding frequencies – of the mC-net of Figure 3.7.

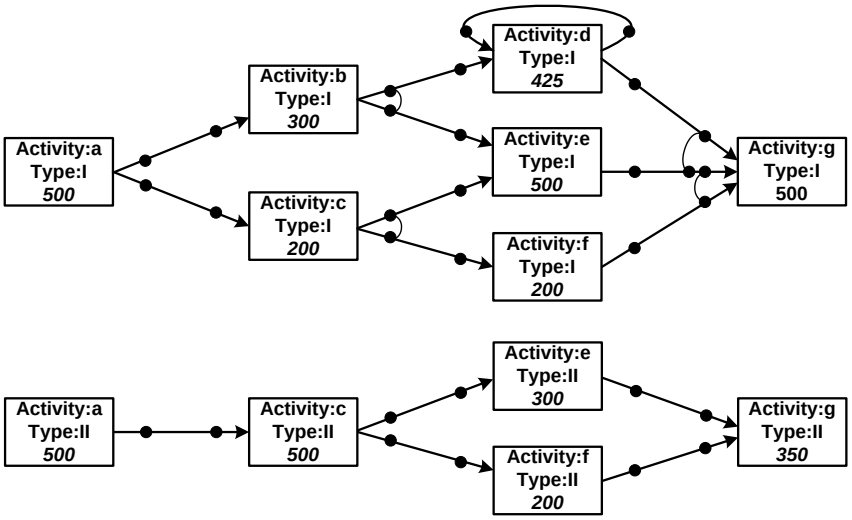


Figure 3.10: A mC-net with 2-D nodes (dimensions *Activity* and *Type*) and 0-D edges.

Sequences of Bindings

A process instance is a sequence of process events (cf. Definition 3.2). These sequences characterize the process execution in terms of executed activities (or tasks) and simple process behavior (i.e., the order in which the process events occurred). Therefore, since complex process behavior such as parallelism is not explicitly represented in a process instance, any kind of analysis directly performed in process instances is quite limited, and usually does not take into account event- or instance-based information.

For traditional process models, an activity binding is defined as the relationship of activated inputs and outputs of a specific activity (cf. Definition 3.4). For multidimensional process models, an event occurrence binding is defined as the – multidimensional – relationship of activated inputs and outputs of a specific event occurrence (cf. Definition 3.10). This means that, in a process instance and given a process model, every process event of the process instance has a relation one-to-one with an activity binding or an event occurrence binding. Hence, by transforming a sequence of process events into a sequence of bindings, it is possible to take into account complex control-flow constructs.

Definition 3.11 (Binding Sequence) *Let a process instance T be a finite sequence of process events $\langle e_1 e_2 ... e_n \rangle$, M a mC-net, and f^{map} a function that, given a mC-net, finds the corresponding event occurrence binding of a process event in T by matching their identifiers. A* binding sequence *is a transformation of T into a sequence of bindings $T' = \langle m_1 m_2 ... m_n \rangle$ by applying f^{map} to every element of T and M.*

The main difference between a sequence of process events and a sequence of bindings is that the first only takes into account event-based information, while the latter also considers workflow-based information. Note that this happens because, unlike sequences of process events, a sequence of bindings is dependent on a specific mC-net (or an aC-net, which can also be seen a particular instance of mC-net).

Figure 3.11 presents a process with parallel behavior represented by a C-net. The

activity bindings of every activity in the C-net are provided in Table 3.10. Table 3.11 shows that there are four distinct sequences of process events that fit in this process. The sequences of activities $\langle ABCDEG\rangle$, $\langle ABCDFG\rangle$, $\langle ACBDEG\rangle$, and $\langle ACBDFG\rangle$ can be used to represent the different sequences of process events. The corresponding – activity – binding sequences are also provided.

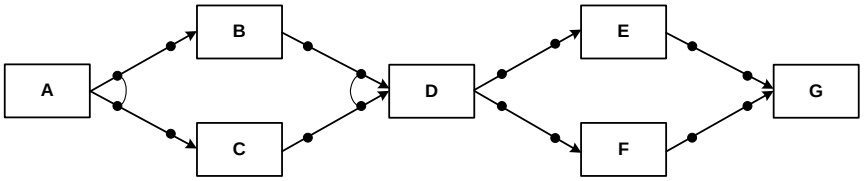


Figure 3.11: A C-net describing a process with parallel behavior.

ID	*Inputs*		*Outputs*	
			B	C
a_1		$\mapsto$	•	•

(a) Bindings of A.

ID	*Inputs*		*Outputs*
	A		D
b_1	•	$\mapsto$	•

(b) Bindings of B.

ID	*Inputs*		*Outputs*
	A		D
c_1	•	$\mapsto$	•

(c) Bindings of C.

ID	*Inputs*			*Outputs*	
	B	C		E	F
d_1	•	•	$\mapsto$	•	
d_2	•	•	$\mapsto$		•

(d) Bindings of D.

ID	*Inputs*		*Outputs*
	D		G
e_1	•	$\mapsto$	•

(e) Bindings of E.

ID	*Inputs*		*Outputs*
	D		G
f_1	•	$\mapsto$	•

(f) Bindings of F.

ID	*Inputs*			*Outputs*
	E	F		
g_1	•		$\mapsto$	
g_2		•	$\mapsto$	

(g) Bindings of G.

Table 3.10: The activity bindings of Figure 3.11's C-Net.

#	*Process Instance*	*Binding Sequence*	*Frequency*
1	$\langle ABCDEG\rangle$	$\langle a_1b_1c_1d_1e_1g_1\rangle$	350
2	$\langle ABCDFG\rangle$	$\langle a_1b_1c_1d_2f_1g_2\rangle$	300
3	$\langle ACBDEG\rangle$	$\langle a_1c_1b_1d_1e_1g_1\rangle$	225
4	$\langle ACBDFG\rangle$	$\langle a_1c_1b_1d_2f_1g_2\rangle$	275

Table 3.11: List of all possible process instances that fit in the process of the Figure 3.11, and corresponding binding sequences.

Considering that activities B and C are executed in parallel (i.e., it does not matter which of these activities is executed first), one may argue that $\langle ABCDEG \rangle$ and $\langle ACBDEG \rangle$ are the same process instance. The same happens to $\langle ABCDFG \rangle$ and $\langle ACBDFG \rangle$. However, these sequence of activities cannot be used to represent such a process behavior (i.e., workflow-based information). Some insight into this aspect can be provided by binding sequences. By transforming these sequences of activities into binding sequences, it is possible to identify parallel behavior in the process. For instance, by translating the activity A into the binding a_1 (the only possible activity binding for A according to Table 3.10), one can easily identify this parallel behavior in all process instances because a_1 defines that, after A, both B and C must be activated.

By adding workflow constraints in the C-net of the Figure 3.11, it is possible to demonstrate that binding sequences can also take into account instance-based information. This means that two similar process instances (i.e., the same sequence of process events) with different characteristics (i.e., different execution properties) can be distinguished. For instance, a specific process instance of the repair process that can handle different types of products.

Figure 3.12 presents a mC-net with an abstract workflow constraint (e.g., the product or service type) of the process of the Figure 3.11. Different colors represent different constraint values. The – event occurrence – bindings of every event occurrence of the mC-net are provided in Table 3.12. Table 3.13 provides details about both sequences of activities and bindings. Remark that the colored bullets are associated to process instances just to help on the identification of the workflow constraint described by the instance.

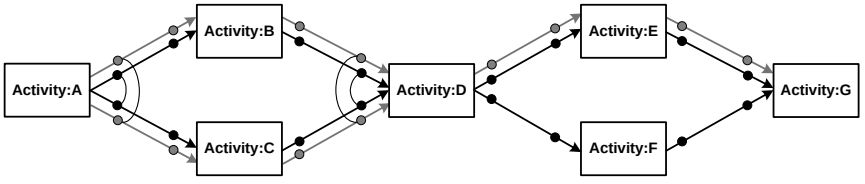


Figure 3.12: A mC-net of the process of the Figure 3.11.

Considering the explicit information provided by sequence of activities, one can conclude that process instances 1a and 1b are represented by $\langle ABCDEG \rangle$. This happens because this kind of sequence does not take into account instance- or workflow-based information. Sequences of – event occurrence – bindings, on the other hand, comprise not only event- but also instance- and workflow-based information. Therefore, process instances with similar behavior but different workflow constraints can be distinguished. For instance, process instances 1a and 1b are represented by distinct binding sequences: $\langle a_1 b_1 c_1 d_1 e_1 g_1 \rangle$ and $\langle a_2 b_2 c_2 d_2 e_2 g_2 \rangle$.

Although binding sequences provide further information than the traditional sequences of process events (or any derivation of them such as sequences of activities), distinguishing process instances with parallel behavior (e.g., 1a and 3a) is not yet possible. This happens because the ordering of bindings is kept in these sequences. A solution for this issue will be discussed in Section 6.3.5.

ID	*Inputs*		*Outputs*	*Frequency*
			B C	
a_1		$\mapsto$	• •	750
a_2		$\mapsto$	• •	400

(a) Bindings of A.

ID	*Inputs*		*Outputs*	*Frequency*
	A		D	
b_1	•	$\mapsto$	•	750
b_2	•	$\mapsto$	•	400

(b) Bindings of B.

ID	*Inputs*		*Outputs*	*Frequency*
	A		D	
c_1	•	$\mapsto$	•	750
c_2	•	$\mapsto$	•	400

(c) Bindings of C.

ID	*Inputs*		*Outputs*	*Frequency*
	B C		E F	
d_1	• •	$\mapsto$	•	175
d_2	• •	$\mapsto$	•	400
d_3	• •	$\mapsto$	•	575

(d) Bindings of D.

ID	*Inputs*		*Outputs*	*Frequency*
	D		G	
e_1	•	$\mapsto$	•	175
e_2	•	$\mapsto$	•	400

(e) Bindings of E.

ID	*Inputs*		*Outputs*	*Frequency*
	D		G	
f_1	•	$\mapsto$	•	575

(f) Bindings of F.

ID	*Inputs*		*Outputs*	*Frequency*
	E F			
g_1	•	$\mapsto$		175
g_2	•	$\mapsto$		400
g_3	•	$\mapsto$		575

(g) Bindings of G.

Table 3.12: The event occurrence bindings of Figure 3.12's mC-Net.

#	*Process Instance*	*Binding Sequence*	*Frequency*
1a	$\langle ABCDEG \rangle^{\bullet}$	$\langle a_1 b_1 c_1 d_1 e_1 g_1 \rangle$	100
1b	$\langle ABCDEG \rangle^{\bullet}$	$\langle a_2 b_2 c_2 d_2 e_2 g_2 \rangle$	250
2a	$\langle ABCDFG \rangle^{\bullet}$	$\langle a_1 b_1 c_1 d_3 f_1 g_3 \rangle$	300
3a	$\langle ACBDEG \rangle^{\bullet}$	$\langle a_1 c_1 b_1 d_1 e_1 g_1 \rangle$	75
3b	$\langle ACBDEG \rangle^{\bullet}$	$\langle a_2 c_2 b_2 d_2 e_2 g_2 \rangle$	150
4a	$\langle ACBDFG \rangle^{\bullet}$	$\langle a_1 c_1 b_1 d_3 f_1 g_3 \rangle$	275

Table 3.13: List of all possible process instances that fit in the process of the Figure 3.12, and corresponding binding sequences.

3.3 Control-Flow Mining

Control-flow mining is particular application of process mining. It consists of the discovery of a process model that reflects the process behavior observed in process data. In this section two control-flow mining techniques based on C-nets are introduced: the Flexible Heuristics Miner and the Multidimensional Heuristics Miner. Both techniques are designed to mine a process model from process data. The difference between these techniques relies on the type of process model produced. The first one produces an aC-net, a one-dimensional process model over the dimension *Activity* (i.e., a traditional process model). The second one produces a mC-net, an n-dimensional process model over n given dimensions present in (or derived from) the process data (i.e., a multidimensional process model). As mC-nets (multidimensional process models) are generalization of aC-nets (traditional process models), the Multidimensional Heuristics Miner can also be considered a generalization of the Flexible Heuristics Miner.

3.3.1 Flexible Heuristics Miner

To construct an aC-net on the basis of process data (an event log or an event stream), the data should be analyzed for causal dependencies [131, 132], e.g., if an activity is always followed by another activity then it is likely that there is a dependency relation between both activities. In order to analyze these relations, some basic relations over the activities in the process data are firstly introduced. The basic relations are used to define dependency measures between activities.

Definition 3.12 (Basic Relations) *Let A be a set of activities. $\delta \in A^*$ is a process instance, $W : A^* \to \mathbb{N}$ is a process dataset[5], and $x, y \in A$:*

1. *$x >_W y$ iff there is a process instance $\delta = \langle a_1 a_2 a_3 \dots a_n \rangle$ and $i \in \{1, \dots, n-1\}$ such that $\delta \in W$ and $a_i = x$ and $a_{i+1} = y$ (direct successor),*
2. *$x \gg_W y$ iff there is a process instance $\delta = \langle a_1 a_2 a_3 \dots a_n \rangle$ and $i \in \{1, \dots, n-2\}$ such that $\delta \in W$ and $a_i = a_{i+2} = x$ and $a_{i+1} = y$ and $x \neq y$ (length-two loops),*
3. *$x \ggg_W y$ iff there is a process instance $\delta = \langle a_1 a_2 a_3 \dots a_n \rangle$ and $i < j$ and $i, j \in \{1, \dots, n\}$ such that $\delta \in W$ and $a_i = x$ and $a_j = y$ (direct or indirect successor).*

[5] A^* is the set of all sequences (i.e., process instances) that are composed of zero or more activities of A. $W : A^* \to \mathbb{N}$ is a function from the elements of A^* to $\mathbb{N}$ (i.e., the number of times an element of A^* appears in the process data). In other words, W is a multiset of process instances.

Mining of the Dependency Graph

The starting point of the Heuristics Miner is the construction of a so-called *dependency graph* (DG). A frequency-based metric is used to indicate the strength of a dependency relation between two process events a and b (notation $a \Rightarrow_W b$). The $\Rightarrow_W$ values between the process events of an event log or an event stream are the basis of the heuristic search for determining the actual dependency relations of the process model.

Definition 3.13 (Dependency Measures) *Let W be a process dataset over A, $x, y \in A$, $|x >_W y|$ the number of times $x >_W y$ occurs in W, and $|x \gg_W y|$ is the number of times $x \gg_W y$ occurs in W.*[6]

$$x \Rightarrow_W y = \begin{cases} \frac{|x>_W y| - |y>_W x|}{|x>_W y| + |y>_W x| + 1} & \text{if } (x \neq y) \\ \\ \frac{|x>_W x|}{|x>_W x| + 1} & \text{otherwise} \end{cases} \tag{3.1}$$

$$x \Rightarrow^2_W y = \frac{|x \gg_W y| + |y \gg_W x|}{|x \gg_W y| + |y \gg_W x| + 1} \tag{3.2}$$

Note that $x \Rightarrow_W y \in [-1, 1]$ ($[0, 1]$ if $x = y$) and $x \Rightarrow^2_W y \in [0, 1]$. $x \Rightarrow_W x$ can be used to identify length-one loops. Length-two loops are identified by the $x \Rightarrow^2_W y$ measurement.

A simple example demonstrates the rationale behind the $x \Rightarrow_W y$ measurement. If we apply Equation 3.1 in the situation that, in 5 process instances out of hundreds, activity a is directly followed by activity b but the other way around never occurs, the value of $a \Rightarrow_W b = 5/6 = 0.833$ indicates that the dependency relation is not highly reliable. Actually, these 5 observations are possibly caused by noise. However, if there are 50 process instances in which a is directly followed by b but the other way around never occurs, the value of $a \Rightarrow_W b = 50/51 = 0.980$ indicates that the dependency relation is highly reliable. If there are 50 process instances in which task a is directly followed by b and the other way around occurs once because of some noise, the value of $a \Rightarrow_W b$ is $49/52 = 0.942$ indicates that is still reliable. A high $a \Rightarrow_W b$ value strongly suggests that there is a dependency relation between activities a and b. Nonetheless, the boundary that identifies whether a dependency relation is reliable or not depends on several factors. The threshold appears sensitive for the amount of noise, the degree of concurrency in the underlying process, and the frequency of the involved activities. The dependency values of Definition 3.13 can be applied in two different ways: with or without the *all-tasks-connected* heuristic.

The underlying intuition in the all-tasks-connected heuristic is that each non-initial activity must have at least one other activity that is its cause, and each non-final activity must have at least one dependent activity. Using this information we can first take *the best* candidates (the ones with the highest $x \Rightarrow_W y$ scores). The advantage of using this heuristic is that many dependency relations are tracked without any influence of any parameter setting.[7] Therefore, in the context of a structured process,

[6] Because the dataset W is a multiset, the same sequence of activities can appear more than once in the data and patterns can appear multiple times in a sequence. If, for instance, the pattern ab appears twice in a sequence (e.g., c**ab**efgc**ab**efh), and this sequence appears three times in W (i.e., W(c**ab**efgc**ab**efh)=3) then these appearances count as 6 in the $|a >_W b|$ measurement.

[7] Remark that the best candidates are always taken independently of the parameter setting. The parameters only influence the amount of extra connections that will appear in the control-flow model.

the result is a relative complete and understandable control-flow model even if there is some noise in the data. However, if we use the all-tasks-connected heuristic in the context of a less-structured process the result is a very complex model with all activities connected by a high number of transitions (i.e., a spaghetti model). In these circumstances, this heuristic can be disabled resulting the execution of the Heuristics Miner in a control-flow model with only the most frequent tasks and behavior. By changing the parameters, it is possible to influence the completeness of the control-flow model. Next, we will firstly introduce the different parameters. Afterwards, some mining results with and without the all-tasks-connected heuristic will be presented.

Four threshold parameters are available in the Heuristics Miner to identify whether or not a dependency relation is reliable:

Dependency threshold (σ_D) defines that a dependency relation from an activity x to another y is reliable if $x \Rightarrow_W y \geq \sigma_D$. The default value of this threshold is $\sigma_D = 0.9$.

Length-one loops threshold (σ_{L1L}) defines that a dependency relation involving a single activity x (i.e., a length-one loop, from x to x) is reliable if $x \Rightarrow_W x \geq \sigma_{L1L}$. The default value of this threshold is $\sigma_{L1L} = 0.9$.

Length-two loops threshold (σ_{L2L}) defines that the pair of dependency relations between activities x and y (i.e., a length-two loop, from x to y and then the other way around) is reliable if $x \Rightarrow^2_W y \geq \sigma_{L2L}$. The default value of this threshold is $\sigma_{L2L} = 0.9$.

Relative-to-best threshold (σ_R) is used to evaluate the dependency relations generated by the all-tasks-connected heuristic. This threshold defines that a dependency relation from an activity x to another y that does not satisfy the dependency threshold (i.e., $x \Rightarrow_W y < \sigma_D$) is reliable if one of the following conditions is satisfied.

- $(x \Rightarrow_W z) - (x \Rightarrow_W y) \leq \sigma_R$, where z is the strongest follower of x (i.e., $\nexists w \in A\ (x \Rightarrow_W w) > (x \Rightarrow_W z)$).
- $(z' \Rightarrow_W y) - (x \Rightarrow_W y) \leq \sigma_R$, where z' is the strongest cause of y (i.e., $\nexists w \in A\ (w \Rightarrow_W y) > (z' \Rightarrow_W y)$).

Basically, this threshold tests whether a rejected dependency relation (by the dependency threshold) is almost as reliable as the best accepted relation. The default value of this threshold is $\sigma_R = 0.05$.

These thresholds define which dependency relations between activities should be part of the dependency graph. Hence, the mining of the dependency graph can be adjusted to deal with different kinds of processes. For example, the loop thresholds (i.e., σ_{L1L} and σ_{L2L}) can be set to 1.0 if it is known that the process does not have any type of loop (or there is no interest to have loops in the process model). In this way, the process model will be built without any loop. The major disadvantage of building the dependency graph in this way is the strong influence of these parameters, especially if there is noise in the process dataset. In these circumstances, the selection of the right parameters (in order to avoid including noise in the results) may be difficult.

The mining of the dependency graph in combination with the all-tasks-connected heuristic is described in Algorithm 1. The all-tasks-connected heuristic is implemented in the algorithm lines 4 through 9. In the lines 9, 10 and 11, the minimal connected

process model is extended with other reliable connections. Remark that, as a heuristic-based algorithm, Algorithm 1 simply provides an approximate solution. The quality of the solution can be improved by adjusting the threshold parameters. For further insight into the parameter selection issue see Subsection 5.4.3.

Algorithm 1: Building the Dependency Graph

Input : The process dataset W, the dependency threshold σ_D, the length-one loops threshold σ_{L1L}, the length-two loops threshold σ_{L2L}, and the relative-to-best threshold σ_R.

Output: The dependency graph for W.

Method

1 $A = \{a \mid \exists_{\delta \in W}[a \in \delta]\}$
`// the set of activities appearing in the dataset`
2 $D_1 = \{(a, a) \in A \times A \mid a \Rightarrow_W a \geq \sigma_{L1L}\}$
`// the set of length-one loops`
3 $D_2 = \{(a, b) \in A \times A \mid (a, a) \notin D_1 \wedge (b, b) \notin D_1 \wedge a \Rightarrow^2_W b \geq \sigma_{L2L}\}$
`// the set of length-two loops`
4 $D_{out} = \{(a, b) \in A \times A \mid a \neq b \wedge \forall_{y \in A}[(a \Rightarrow_W b \geq a \Rightarrow_W y) \wedge (a \Rightarrow_W b > 0)]\}$
`// for each activity, the strongest follower`
5 $D_{in} = \{(a, b) \in A \times A \mid a \neq b \wedge \forall_{x \in A}[(a \Rightarrow_W b \geq x \Rightarrow_W b) \wedge (a \Rightarrow_W b > 0)]\}$
`// for each activity, the strongest cause`
6 $D'_{out} = \{(a, y) \in D_{out} \mid a \Rightarrow_W y < \sigma_D \wedge \exists_{(b,z) \in D_{out}}[(a, b) \in D_2 \wedge (b \Rightarrow_W z) - (a \Rightarrow_W y) > \sigma_R]\}$
`// only one follower activity is necessary for a length-two loop`
7 $D_{out} = D_{out} - D'_{out}$
8 $D'_{in} = \{(x, a) \in D_{in} \mid x \Rightarrow_W a < \sigma_D \wedge \exists_{(z,b) \in D_{in}}[(a, b) \in D_2 \wedge (z \Rightarrow_W b) - (x \Rightarrow_W a) > \sigma_R]\}$
`// only one cause activity is necessary for a length-two loop`
9 $D_{in} = D_{in} - D'_{in}$
10 $D''_{out} = \{(a, b) \in A \times A \mid a \Rightarrow_W b \geq \sigma_D \vee \exists_{(a,y) \in D_{out}}[(a \Rightarrow_W y) - (a \Rightarrow_W b) \leq \sigma_R]\}$
11 $D''_{in} = \{(a, b) \in A \times A \mid a \Rightarrow_W b \geq \sigma_D \vee \exists_{(x,b) \in D_{in}}[(x \Rightarrow_W b) - (a \Rightarrow_W b) \leq \sigma_R]\}$
12 **return** $D_1 \cup D_2 \cup D''_{out} \cup D''_{in}$

The process model as depicted in Figure 3.13 is used as an illustrative example to explain the process of building a dependency graph. Let us assume that this model is also used for generating an artificial event log with 1000 random traces. However, during the generation of the event log, the *hidden activities* $D1$, $D2$ and $D3$ are not registered. Hidden activities are a standard solution within the Petri net formalism to deal with more complex and flexible split/join constructs. Moreover, the patterns in which the activity L and the hidden activity $D2$ are involved are *low frequent patterns* (i.e., activity J followed by C, and D followed by K without F as intermediate). The combination of parallelism (after activity A two parallel processes are started), loops (length-one, length-two and longer loops), hidden activities, and low frequent behavior, makes this event log difficult to mine.

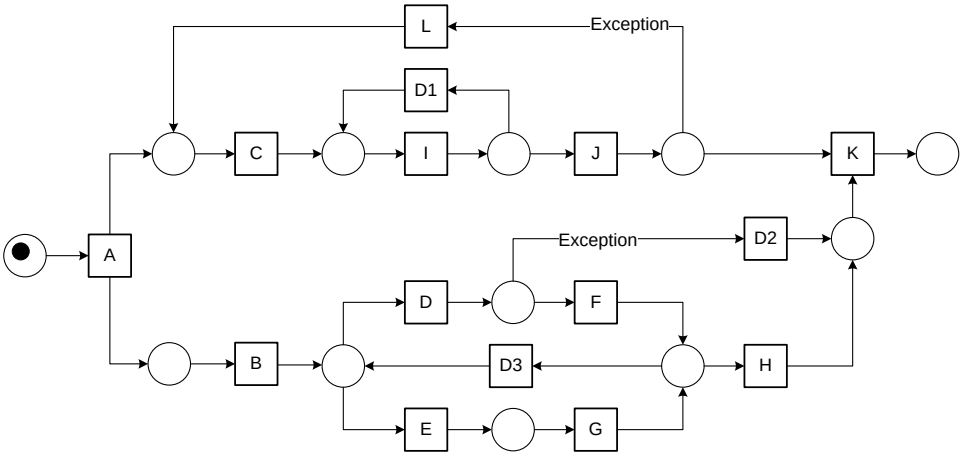


Figure 3.13: The reference process model.

The basic information needed to build the dependency graph based on Figure 3.13 is presented in Table 3.14 (the counting of the direct successors (i.e., $x >_W y$)), Table 3.15 (the counting of the length-two loops (i.e., $x \gg_W y$)), and Table 3.16 (the result of applying Equation 3.1 on Table 3.14).

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	520	480	0	0	0	0	0	0	0	0	0
B	0	0	360	182	198	0	0	0	233	27	0	0
C	0	338	0	125	128	40	48	8	349	0	0	0
D	0	0	63	0	0	586	0	0	193	68	5	6
E	0	0	73	0	0	0	619	0	236	67	0	3
F	0	0	16	124	134	0	0	327	212	88	0	7
G	0	0	16	143	145	0	0	359	220	105	0	10
H	0	0	11	0	0	0	0	0	252	105	614	5
I	0	119	0	209	236	179	210	166	315	576	0	0
J	0	23	0	135	155	102	117	118	0	0	381	5
K	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	17	3	2	1	4	9	0	0	0	0

Table 3.14: Direct successor counting ($x >_W y$-counting for activities).

The value 520 in position (A,B) indicates that A was followed by B 520 times in the event log. Analogously, the value 315 in position (I,I) indicates that I followed itself 315 times. This value clearly suggests that there is an length-one loop on I.

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	89	0	0	0	0	0	0
E	0	0	0	0	0	0	104	0	0	0	0	0
F	0	0	0	110	0	0	0	0	0	0	0	0
G	0	0	0	0	133	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0	0	0	0	0
I	0	19	0	40	63	59	57	97	116	0	0	0
J	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.15: Length-two loops counting ($x \gg_W y$-counting for activities).

The value 89 in position (D,F) indicates that there are 89 DFD patterns in the event log. Remark the high value between the length one loop-activity I and many other activities (i.e., B, D, E F, G and H). This is caused by the looping behavior of I in combination with the parallel behavior of the other mentioned activities.

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	**.998**	**.998**	0	0	0	0	0	0	0	0	0
B	-.998	0	.031	**.995**	**.995**	0	0	0	.323	.084	0	0
C	-.997	-.031	0	.328	.272	.421	.492	-.150	**.997**	0	0	-.944
D	0	-.995	-.328	0	0	**.650**	-.993	0	-.040	-.328	**.833**	.300
E	0	-.995	-.272	0	0	-.993	**.620**	0	0	-.395	0	.167
F	0	0	-.421	-.650	*.993*	0	0	**.997**	.0842	-.073	0	.667
G	0	0	-.492	*.993*	-.620	0	0	**.997**	.0232	-.054	0	.400
H	0	0	.15	0	0	-.997	-.997	0	.205	-.058	**.998**	-.267
I	0	-.323	-.997	.040	0	-.084	-.023	-.205	*.997*	**.998**	0	0
J	0	-.078	0	.328	.395	.073	.054	.058	-.998	0	**.997**	**.833**
K	0	0	0	-.833	0	0	0	-.998	0	-.997	0	0
L	0	0	**.944**	-.300	-.167	-.667	-.400	.267	0	-.833	0	0

Table 3.16: All $x \Rightarrow_W y$-values.

According to lines 4 and 5 of Algorithm 1, the best candidate (bold face) of each row and column is accepted. However, based on lines 6 and 7, in case of length-two loops (i.e., DF and EG) the weakest candidates (underlined) are removed (these loops only need one (extra) incoming and one outgoing candidate, to be part of the dependency graph). Extra connections may be accepted depending on the values of the parameters (i.e., lines 2, 3, 8 and 9). These connections are displayed in *Italics* (e.g., from G to D). Considering the information in Table 3.16 and assuming the thresholds default values (i.e., $\sigma_D = \sigma_{L1L} = \sigma_{L2L} = 0.9$ and $\sigma_R = 0.05$), each step of the process of building the dependency graph (i.e., each line of Algorithm 1) can be described as follows.

1. The first step of the algorithm is the construction of the set A (the set of all activities appearing in the log).
2. Looking at the diagonal of Table 3.14 there is only one candidate for D_1: activity I is 315 times followed by itself. The value of $I \Rightarrow_W I = 315/(315+1) \geq \sigma_{L1L}$, resulting in $D_1 = \{(I, I)\}$.
3. For this step of the algorithm we make use of Table 3.15. The table indicates that the subsequence $\langle DFD \rangle$ appears 89 in the log and subsequence $\langle FDF \rangle$ 110 times. Therefore $D \Rightarrow^2_W F = (89+110)/(89+110+1) = 0.995$. Because $(F, F) \notin D_1$ and $(D, D) \notin D_1$ and $0.995 \geq \sigma_{L2L}$ both $(F, D) \in D_2$ and $(D, F) \in D_2$. The same argumentation counts for the case (E, G) resulting in $D_2 = \{(F, D), (D, F), (E, G), (G, E)\}$.
4. Based on Table 3.16, check each row for the highest value (the strongest follower). For example, for activity C the highest value (in boldface) is 0.997; therefore (C, I) is in the set D_{out}.
5. Based on Table 3.16, check each column for the highest value (the strongest cause). For example, for activity K the highest value (in boldface) is 0.998; therefore (H, K) is in the set D_{in}.

6,7. The relation between activities D and F can be used to explain this step. They are in a direct loop (i.e., $(D, F) \in D_2$). The strongest output connection of D beside F is K (0.833), and from F is H (0.997). For this reason $(D, F) \in D'_{out}$ (is

not strictly necessary) and will be removed from D_{out} (line 7 of the algorithm). In Table 3.16 the removed connections are marked with underlining.

8,9. Analogue to lines 6 and 7, but now for the incoming connections.

10,11. Depending on the values of the parameters, extra connections are accepted if both the absolute dependency threshold σ_D **and** the relative-to-best threshold σ_R are satisfied. Remark that for the default parameter setting the dependency relation between D and K is not accepted because $D \Rightarrow_W K = 0.333 < 0.9$ (Table 3.16). However, the connection from J to L is accepted, because the all tasks connected heuristic is active. In the matrix of Table 3.16 the extra accepted dependency values are displayed in *Italics*.

12. Finally, the information in the different sets is combined to perform the last step of the algorithm.

Inputs	*Activity*	*Outputs*
$\emptyset$	A	$\{B, C\}$
$\{A\}$	B	$\{D, E\}$
$\{A, L\}$	C	$\{I\}$
$\{B, F, G\}$	D	$\{F\}$
$\{B, F, G\}$	E	$\{G\}$
$\{D\}$	F	$\{D, E, H\}$
$\{E\}$	G	$\{D, E, H\}$
$\{F, G\}$	H	$\{K\}$
$\{C, I\}$	I	$\{I, J\}$
$\{I\}$	J	$\{K, L\}$
$\{J, H\}$	K	$\emptyset$
$\{J\}$	L	$\{C\}$

Table 3.17: The resulting dependency graph.

Remark that low frequent connections are unlikely to be part of the dependency graph if

- the thresholds σ_D, σ_{L1L}, and σ_{L2L} are near to one,
- the threshold σ_R is near to zero, and
- the all-tasks-connected heuristic is not applied.

The dependency graph provides information about the dependency between activities, but the types of splits/joins are not yet mined.

Mining of the Splits and Joins

The second step of the Flexible Heuristics Miner is the identification of output and input bindings (or splits and joins) in the dependency graph. For each activity in the dependency graph, all the possible input and output bindings are derived from parsing the process data on the dependency graph. Applying the aC-net definition (cf. Definition 3.6), each activity is characterized by two multisets of – input and output – bindings and corresponding frequencies. Table 3.18 summarizes the possible input and output bindings in the dependency graph. Remark that these results are based on

the artificial event log with 1000 random traces in combination with the dependency graph of Table 3.17.

Input Bindings	*Activity*	*Output Bindings*
$[\ \emptyset^{1000}\ ]$	A	$[\ \{B,C\}^{1000}\ ]$
$[\ \{A\}^{1000}\ ]$	B	$[\ \{D\}^{467}, \{E\}^{533}\ ]$
$[\ \{A\}^{1000}, \{L\}^{36}\ ]$	C	$[\ \{I\}^{1036}\ ]$
$[\ \{B\}^{467}, \{F\}^{222}, \{G\}^{232}\ ]$	D	$[\ \{F\}^{908}\ ]$
$[\ \{B\}^{533}, \{F\}^{215}, \{G\}^{250}\ ]$	E	$[\ \{G\}^{998}\ ]$
$[\ \{D\}^{908}\ ]$	F	$[\ \{D\}^{222}, \{E\}^{215}, \{H\}^{471}\ ]$
$[\ \{E\}^{998}\ ]$	G	$[\ \{D\}^{232}, \{E\}^{250}, \{H\}^{516}\ ]$
$[\ \{F\}^{471}, \{G\}^{516}\ ]$	H	$[\ \{K\}^{987}\ ]$
$[\ \{C\}^{1036}, \{I\}^{974}\ ]$	I	$[\ \{I\}^{974}, \{J\}^{1036}\ ]$
$[\ \{I\}^{1036}\ ]$	J	$[\ \{K\}^{1000}, \{L\}^{36}\ ]$
$[\ \{J,H\}^{987}, \{J\}^{13}\ ]$	K	$[\ \emptyset^{1000}\ ]$
$[\ \{J\}^{36}\ ]$	L	$[\ \{C\}^{36}\ ]$

Table 3.18: All input and output bindings – and corresponding frequencies – of the dependency graph of Table 3.17.

Figure 3.14 depicts the aC-net based on the input and output bindings of Table 3.18. Note that, following the aC-net definition, the numerical information in nodes represent the activity frequency. Analogously, that information on edges represent the number of times a transition from an activity to another occurred.

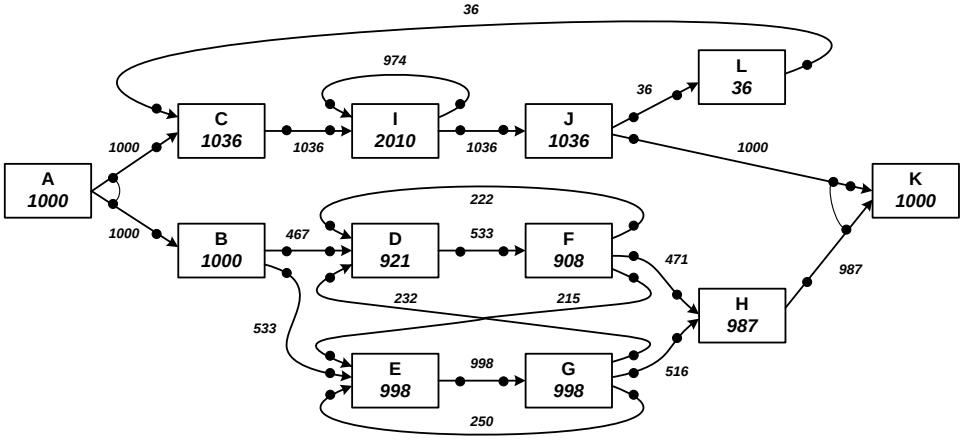


Figure 3.14: The resulting aC-net.

The process of mining the splits and joins of a dependency graph is defined in Algorithm 2. Remark that the strategy for computing splits and joins described in this algorithm may not provide the correct results for some exceptional cases. No inconsistent results (incorrect splits or joins) were found in our experiments. However, a further assessment is necessary to find the limitations of the strategy.

The dependency graph of Table 3.17 is used as an illustrative example to explain the mining the splits and joins. Let us consider $\langle ABDCIFIJEGHK \rangle$ as the process

Algorithm 2: Computing the Splits and Joins

Input : The process dataset W and the dependency graph DG.

Output: The multisets of – input and output – bindings and corresponding frequencies.

Method

1 $splits \leftarrow \emptyset, joins \leftarrow \emptyset$; `// the multisets of -- input and output -- bindings and corresponding frequencies`
2 **foreach** *process instance* $\delta \in W$ **do**
3 **for** $i \leftarrow 1$ **to** *length of* δ **do** $B_i^I \leftarrow \emptyset$, $B_i^O \leftarrow \emptyset$;
4 **for** $i \leftarrow 1$ **to** *length of* δ **do**
5 $a_i \leftarrow$ the activity of the process event in the i^{th} position of δ;
6 $O_i \leftarrow$ the set of outputs of a_i in DG ; `// searching activated outputs of` a_i
7 $visitedEvents \leftarrow \emptyset$;
8 **for** $j \leftarrow (i+1)$ **to** *length of* δ **do**
9 $a_j \leftarrow$ the activity of the process event in the i^{th} position of δ;
10 $I_j \leftarrow$ the set of inputs of a_j in DG;
11 **if** $a_j \in O_i \wedge \nexists_{x \in visitedEvents}[x \in I_j]$ **then**
12 $B_i^O \leftarrow B_i^O \cup \{a_j\}$, $B_j^I \leftarrow B_j^I \cup \{a_i\}$;
end
13 **if** $a_j = a_i$ **then** $j \leftarrow$ length of δ **else** $visitedEvents \leftarrow visitedEvents \cup \{a_j\}$;
end
14 $I_i \leftarrow$ the set of inputs of a_i in DG ; `// searching activated inputs of` a_i
15 $visitedEvents \leftarrow \emptyset$;
16 **for** $j \leftarrow (i-1)$ **to** 1 **do**
17 $a_j \leftarrow$ the activity of the process event in the i^{th} position of δ;
18 $O_j \leftarrow$ the set of outputs of a_j in DG;
19 **if** $a_j \in I_i \wedge \nexists_{x \in visitedEvents}[x \in O_j]$ **then**
20 $B_i^I \leftarrow B_i^I \cup \{a_j\}$, $B_j^O \leftarrow B_j^O \cup \{a_i\}$;
end
21 **if** $a_j = a_i$ **then** $j \leftarrow 1$ **else** $visitedEvents \leftarrow visitedEvents \cup \{a_j\}$;
end
end
22 **for** $i \leftarrow 1$ **to** *length of* δ **do**
23 $a_i \leftarrow$ the activity of the process event in the i^{th} position of δ;
24 $splits \leftarrow splits \cup \{B_i^O\}^{a_i}$, $joins \leftarrow joins \cup \{B_i^I\}^{a_i}$;
end
end
25 **return** ($joins$,$splits$);

instance being parsed. The mining of the output binding (split) of the activity A can be described as follows (lines 4-13). According to Table 3.17, the outputs of A are B and C. Therefore, four output bindings are possible:

- $\emptyset$, if A is not followed by any of its outputs. This output binding can occur when A is an end-activity or there is some noise in the data.
- $\{B\}$, if A is followed exclusively by B. This output binding can be considered as an XOR-Split.
- $\{C\}$, if A is followed exclusively by C. This output binding can be considered as an XOR-Split.
- $\{B, C\}$, if A is followed by both B and C. This output binding can be considered as an AND-Split.

Note that the activity A appears in the first position of the process instance (i.e., $i = 1$ in line 4 and a_1 in line 5). Hence, the output binding of this activity is defined by the subsequence of activities from the instance's second position onwards (i.e., $j = 2$ to 12 in the loop of line 8). An activity x in this subsequence is part of the output binding of A if (line 12):

- x is part of the set of outputs of A (i.e., set $O_1 = \{B, C\}$ in line 6), and
- there is no other activity y in between of A and x (i.e., set *visitedEvents* in lines 7, 11 and 13) such that y is part of the set of inputs of x. This condition checks if this possible output is actually activated by A and not by any other activity.

Activities B and C (a_2 and a_4 in line 9) satisfy these conditions. $B \in O_1$ and there is not any activity in between of A and B. $C \in O_1$ and none of the activities in between of A and C (i.e., $visitedEvents = \{B, D\}$) is part of the inputs of C (i.e., B and D are not in $I_4 = \{A, L\}$). Therefore, the output binding of A in the given process instance is defined as an AND-Split $\{B, C\}$. Consequently, A is added to the input bindings of B and C (line 12). Finally, the value $\{B, C\}^A$ is added into the multiset of output bindings of the dependency graph (line 24).

The mining of joins is analogous to the process described above. The mining of the input binding (join) of the activity E can be described as follows (lines 4-5 and 14-21). According to Table 3.17, the inputs of E are B, F and G. Therefore, for E, eight input bindings are possible: $\emptyset$, $\{B\}$, $\{F\}$, $\{G\}$, $\{B, F\}$, $\{B, G\}$, $\{F, G\}$, and $\{B, F, G\}$. Note that the activity E appears in the ninth position of the process instance (i.e., $i = 9$ in line 4 and a_9 in line 5). Hence, the input binding of this activity is defined by the subsequence of activities from the instance's eighth position backwards (i.e., $j = 8$ to 1 in the loop of line 16). An activity x in this subsequence is part of the input binding of E if (line 20):

- x is part of the set of inputs of E (i.e., set $I_9 = \{B, F, G\}$ in line 14), and
- there is no other activity y in between of x and A (i.e., set *visitedEvents* in lines 15, 19 and 21) such that y is part of the set of outputs of x. Analogously to the mining of splits, this condition checks if this possible input is actually activated by E and not by any other activity.

Although activities B and F appear in the subsequence (a_2 and a_6 in line 17), only F satisfies these conditions. $F \in I_9$ and none of the activities in between of F and E (i.e., $D \in visitedEvents = \{I, J\}$) is part of the outputs of F (i.e., I and J are not in $O_6 = \{D, E, H\}$). $B \in I_9$, but activity D appears in between of B and E (i.e., $D \in visitedEvents = \{D, C, I, F, J\}$), and D is part of the set of outputs of B (i.e., $D \in O_2 = \{D, E\}$). Therefore, the input binding of E in the given process instance is defined as an XOR-Join $\{F\}$. Consequently, E is added to the output binding of F

(line 20). Finally, the value $\{F\}^E$ is added into the multiset of input bindings of the dependency graph (line 24).

Each activity of a process instance is characterized by an input and an output binding (i.e., a join and a split). The multisets of all – input and output – bindings derived from parsing the process data on the dependency graph define the aC-net (e.g., Table 3.18 and Figure 3.14).

Mining of the Long-Distance Dependencies

The final step of the Flexible Heuristics Miner is the identification of dependencies that are not represented yet in the dependency graph. Called long-distance dependencies (or non-free choice), these relations indicate cases in which an activity X depends indirectly on another activity Y to be executed. That means that, in a split or a join point, the choice may depend on choices made in other parts of the process model. Figure 3.15 depicts a Petri Net with two distinct long-distance dependencies (i.e., the relations $B \Rightarrow E$ and $C \Rightarrow F$). Note that, in this example, there are only two possible sequences: $\langle ABDEG \rangle$ and $\langle ACDFG \rangle$. However, without mining the long-distance dependencies, the dependency graph does not ensure that activity E is only executed if D follows B. The same happens for F. Thus, non-valid sequences such as $\langle A\underline{B}D\underline{F}G \rangle$ or $\langle A\underline{C}D\underline{E}G \rangle$ might fit in the process model. Figure 3.16 shows the corresponding C-nets with and without mining the long-distance dependencies.

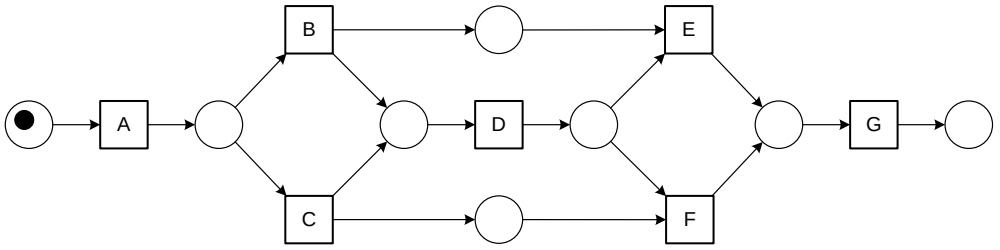


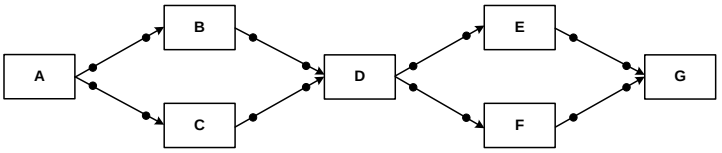
Figure 3.15: A Petri net with long-distance dependencies.

In order to handle the long-distance dependency issue, a new frequency-based metric is defined. Basically, this metric takes into account the indirect relation between activities (i.e., the direct or indirect successor counter of Definition 3.12). The main idea is to find pairs of activities with similar frequencies in which the first activity is directly or indirectly followed by the second one. These cases are measured through the $x \Rightarrow^l_W y$ measure (cf. Definition 3.14). Every pair with a non-zero $\Rightarrow^l_W$-value is designated as a long-dependency relation.

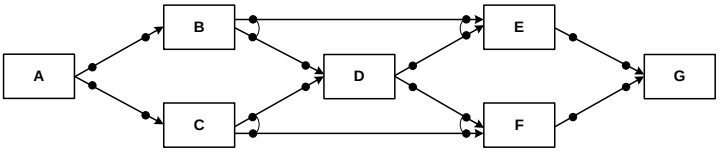
Definition 3.14 (Long-Distance Dependency Measure) *Let A be a set of activities, W a process dataset over A, $x, y \in A$, $|x \ggg_W y|$ the number of times $x \ggg_W y$ occurs in W[8], and $|x|$ is the number of times x occurs in W.*

$$x \Rightarrow^l_W y = \frac{2 \times (|x \ggg_W y|)}{|x| + |y| + 1} - \frac{2 \times abs(|x| - |y|)}{|x| + |y| + 1} \tag{3.3}$$

[8] In the pattern cde**a**f**g**h**b**i**b**j**k**a**l**a**d**e**f**b**g**h only the underlined appearances of the pattern a...b contribute to the value $|a \overline{\ggg_W} b|$ (i.e., only a...b patterns without other a's or b's between them).



(a) C-net without long-distance dependency relations.



(b) C-net with long-distance dependency relations.

Figure 3.16: The Figure 3.15's corresponding C-nets with and without long-distance dependency relations.

Note that $x \Rightarrow^{l}_{W} y \in [0, 1]$.

A long-distance relation from an activity x *to another* y ($x \Rightarrow^{l}_{W} y$) *is considered reliable if* $x \Rightarrow^{l}_{W} y \geq \sigma_L$, *where* σ_L *is the* ***long-distance threshold****. The default value of this threshold is* $\sigma_L = 0.9$.

A $x \Rightarrow^{l}_{W} y$ value close to 1 of the first part of the expression indicates that activity x is always followed by activity y. A value close to 0 of the second part indicates that the frequency of activities x and y is about equal. That means that an overall value close to 1 indicates both: activity x is always followed by activity y and the frequencies of activities x and y are about equal. Remark that some long-dependency cases are already implicitly represented in the dependency graph. A good example is the relation $A \Rightarrow D$ in Figure 3.15, which its long-distance dependency value is close to 1.0 but no extra dependency relation is necessary. This happens because A is always indirectly followed by D, turning redundant the extra direct connection from A to D. Taking into account this remark, it is finally defined that a long-dependency relation $x \Rightarrow y$ (with $x, y \in A$) should be part of the dependency graph whenever the relation satisfies the long-distance threshold and it is possible to go from x to an end task without visiting y.

The process of mining the long-distance dependencies in a dependency graph is described in Algorithm 3. This process starts by determining the set of long-distance relations (D) that satisfy the long-distance threshold (lines 1 and 2). Every relation in D should not be in the dependency graph. For each relation (a, b) in D (line 3), function *escapeToEnd* checks whether it is possible to go from a to an end task without visiting b. If so, the long-distance relation (a, b) is added into the dependency graph (line 4).

Note that every time a new long-distance dependency relation is added into the dependency graph the relation activities' inputs and outputs change as well as their input and output bindings. So, at the end of this stage (i.e., after adding all long-

Algorithm 3: Computing the Long-Distance Dependencies

Input : The process dataset W, the dependency graph DG, the multiset of output bindings (splits) and corresponding frequencies S, and the long-distance threshold σ_L.

Output: The dependency graph with long-distance dependencies for W.

Method

1 A is the set of activities appearing in the dependency graph;
2 $D = \{(a,b) \in A \times A \mid (a,b) \notin DG \wedge a \Rightarrow_W^l b \geq \sigma_L\}$; `// the set of potential long-distance relations`
3 **foreach** *pair of activities* $(a,b) \in D$ **do**
4 | **if** `escapeToEnd(`a`,`S`,{`b`})` $= true$ **then** $DG \leftarrow DG \cup \{(a,b)\}$;
end
5 **return** DG;

function escapeToEnd($currentActivity$, S, $visitedActivities$)
if $currentActivity \in visitedActivities$ **then return** false;
$X \leftarrow \{A \mid \exists_{A^b \in S}[b = currentActivity]\}$; `// the output bindings of activity` $currentActivity$
if $\emptyset \in X$ **then return** true ; `// tests whether the activity` $currentActivity$ `is an end activity`
$visitedActivities \leftarrow visitedActivities \cup \{currentActivity\}$;
return $\bigvee_{B \in X} \bigwedge_{b \in B}$ `escapeToEnd(`b`,`S`,`$visitedActivities$`)`;

distance dependencies into the dependency graph), it is necessary to recompute the split/join information.

3.3.2 Multidimensional Heuristics Miner

The Multidimensional Heuristics Miner can be seen as a generalization of the Flexible Heuristics Miner. Instead of one-dimensional process models (over the dimension *Activity*), this algorithm can build process models over multiple dimensions, the so-called multidimensional process models. The main differences between the Multidimensional and the Flexible Heuristics Miner algorithms are basically described as follows.

Mining of the dependency graph follows the same approach as in the Flexible Heuristics Miner, but over event occurrences instead of activities. Additionally, instead of using abstract dependency relations, workflow constraints are used to characterize the transitions between event occurrences.

Mining of the splits and joins is based on the approach considered in the Flexible Heuristics Miner. However, instead of input and output bindings, this step is redesigned to find event occurrence bindings (cf. Definition 3.10). Remark that event occurrence bindings are formed by input and output bindings characterized by workflow constraints.

Mining of the long-distance dependencies is not considered in the Multidimensional Heuristics Miner. Eventually, long-distance dependencies may be identified

by *process patterns*, which can be derived from a mC-net. Process patterns are introduced in Chapter 6.

Like for aC-nets, to construct a mC-net on the basis of process data (an event log or an event stream), the data should be analyzed for causal dependencies between event occurrences. These multidimensional relations are a generalization of the basic ones used in the Flexible Heuristics Miner.

Definition 3.15 (Basic Multidimensional Relations) *Let E be a set of event occurrences. $\delta \in E^*$ is a process instance, $W : E^* \to \mathbb{N}$ is a process dataset[9], and $x, y \in E$:*

1. *$x >_W y$ iff there is a process instance $\delta = \langle e_1 e_2 e_3 \dots e_n \rangle$ and $i \in \{1, \dots, n-1\}$ such that $\delta \in W$ and $e_i = x$ and $e_{i+1} = y$ (direct successor),*
2. *$x \gg_W y$ iff there is a process instance $\delta = \langle e_1 e_2 e_3 \dots e_n \rangle$ and $i \in \{1, \dots, n-2\}$ such that $\delta \in W$ and $e_i = e_{i+2} = x$ and $e_{i+1} = y$ and $x \neq y$ (length-two loops),*

Remark that this definition and Definition 3.12 are equivalent if the event occurrences in E are characterized exclusively by the dimension Activity.

Mining of the Multidimensional Dependency Graph

Like for any other Heuristics Miner-based algorithm, the construction of the dependency graph is the starting point for mining a process model. However, for building a multidimensional process model, the traditional dependency graphs need to be adapted to support multiple dimensions. Therefore, instead of describing activities and their – abstract – dependency relations, the dependency graphs should be extended in such a way that event occurrences and their constrained dependency relations can be described. These extended dependency graphs are designated as *multidimensional dependency graphs*.

The process of building a multidimensional dependency graph relies on the same frequency-based metrics $\Rightarrow_W$ and $\Rightarrow^2_W$ used in the Flexible Heuristics Miner (cf. Definition 3.13). However, instead of applying the measures over activities, the dependency measures are applied over event occurrences. This means that, being x and y event occurrences, the values $x >_W y$ and $x \gg_W y$ – needed to compute the dependency values $x \Rightarrow_W y$ and $x \Rightarrow^2_W y$ – are determined according to Definition 3.15.

The Multidimensional Heuristics Miner works with the same parameters as the Flexible Heuristics Miner. The mining of the multidimensional dependency graph in combination with the all-tasks-connected heuristic is described in Algorithm 4. Remark that this algorithm is a generalization of Algorithm 1 (page 64). For an easier understanding, the same line numbering is preserved in this algorithm (i.e., the same operations are identified by the same line numbers in both algorithms 1 and 4).

The one-dimensional sub-process as depicted in Figure 3.17 is used as an illustrative example to explain the process of building a multidimensional dependency graph. Let us assume that the process data from which the model is generated contains information about not only activities but also resources (originators) and product types. For this sub-process, details about these dimensions are provided in Table 3.19.

[9] E^* is the set of all sequences (i.e., process instances) that are composed of zero or more event occurrences of E. $W : E^* \to \mathbb{N}$ is a function from the elements of E^* to $\mathbb{N}$ (i.e., the number of times an element of E^* appears in the process data). In other words, W is a multiset of process instances.

Algorithm 4: Building the Multidimensional Dependency Graph

Input : The process dataset W, the set of dimensions by which event occurrences are characterized (D^E), the set of dimensions by which dependency relations are characterized (D^W), the dependency threshold σ_D, the length-one loops threshold σ_{L1L}, the length-two loops threshold σ_{L2L}, and the relative-to-best threshold σ_R.

Output: The multidimensional dependency graph for W.

Method

1 $E' = \{D{:}d \mid \exists_{\delta \in W}[\exists_{e \in \delta}[d \text{ is the value of dimension } D \text{ in } e]] \wedge D \in D^E \cup D^W\}$
$E = \{X \in \mathcal{P}(E') \mid \forall_{D_1:d_1 \in X} \nexists_{D_2:d_2 \in X}[D_1 = D_2 \wedge d_1 \neq d_2]\}$
// the set of event occurrences appearing in the dataset

2 $D_1 = \{(a,a) \in E \times E \mid a \Rightarrow_W a \geq \sigma_{L1L}\}$
// the set of length-one loops

3 $D_2 = \{(a,b) \in E \times E \mid (a,a) \notin D_1 \wedge (b,b) \notin D_1 \wedge a \Rightarrow_{2\ W} b \geq \sigma_{L2L}\}$
// the set of length-two loops

4 $D_{out} = \{(a,b) \in E \times E \mid a \neq b \wedge \forall_{y \in A}[(a \Rightarrow_W b \geq a \Rightarrow_W y) \wedge (a \Rightarrow_W b > 0)]\}$
// for each activity, the strongest follower

5 $D_{in} = \{(a,b) \in E \times E \mid a \neq b \wedge \forall_{x \in A}[(a \Rightarrow_W b \geq x \Rightarrow_W b) \wedge (a \Rightarrow_W b > 0)]\}$
// for each activity, the strongest cause

6 $D'_{out} = \{(a,y) \in D_{out} \mid a \Rightarrow_W y < \sigma_D \wedge \exists_{(b,z) \in D_{out}}[(a,b) \in D_2 \wedge (b \Rightarrow_W z) - (a \Rightarrow_W y) > \sigma_R]\}$
// only one follower activity is necessary for a length-two loop

7 $D_{out} = D_{out} - D'_{out}$

8 $D'_{in} = \{(x,a) \in D_{in} \mid x \Rightarrow_W a < \sigma_D \wedge \exists_{(z,b) \in D_{in}}[(a,b) \in D_2 \wedge (z \Rightarrow_W b) - (x \Rightarrow_W a) > \sigma_R]\}$
// only one cause activity is necessary for a length-two loop

9 $D_{in} = D_{in} - D'_{in}$

10 $D''_{out} = \{(a,b) \in E \times E \mid a \Rightarrow_W b \geq \sigma_D \vee \exists_{(a,y) \in D_{out}}[(a \Rightarrow_W y) - (a \Rightarrow_W b) \leq \sigma_R]\}$

11 $D''_{in} = \{(a,b) \in E \times E \mid a \Rightarrow_W b \geq \sigma_D \vee \exists_{(x,b) \in D_{in}}[(x \Rightarrow_W b) - (a \Rightarrow_W b) \leq \sigma_R]\}$

12 $X \leftarrow D_1 \cup D_2 \cup D''_{out} \cup D''_{in}$

return `split(`X,D^E,D^W`)`

function split(X, D^E, D^W)
constrainedPairs $\leftarrow \emptyset$
foreach *pair of event occurrences* $(a,b) \in X$ **do**
$x \leftarrow \{D{:}d \mid D{:}d \in a \wedge D \in D^E\}$ // the event constraint of a
$x' \leftarrow \{D{:}d \mid D{:}d \in a \wedge D \in D^W\}$ // the workflow constraint of a
$y \leftarrow \{D{:}d \mid D{:}d \in b \wedge D \in D^E\}$ // the event constraint of b
$y' \leftarrow \{D{:}d \mid D{:}d \in b \wedge D \in D^W\}$ // the workflow constraint of b
$z \leftarrow \{D{:}d_1 \rightarrow d_2 \mid D{:}d1 \in x' \wedge D{:}d2 \in y'\}$ // the workflow constraint of the transition (a,b)
constrainedPairs $\leftarrow$ *constrainedPairs* $\cup \{(x,y,z)\}$
end
return *constrainedPairs*

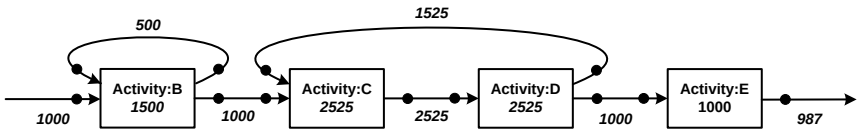


Figure 3.17: An example of a one-dimensional sub-process.

Dimension	*Dimension Values*
Activity	$\{B, C, D, E\}$
Resource	$\{X, Y, Z\}$
Type	$\{I, II\}$

Table 3.19: Characterization of the dimensions in the process data from which the model of Figure 3.17 is generated.

Considering the dimensions *Activity*, *Resource*, and *Type*, eight different three-dimensional models of the same sub-process (i.e., different perspectives of the sub-process) can be built:

- **3-D nodes + 0-D edges**
  1. The three dimensions (*Activity*, *Resource*, and *Type*) are used to define the model's event occurrences. In this case, the dependency relations between event occurrences are not constrained (i.e., there are no workflow constraints).
- **2-D nodes + 1-D edges**
  2. The dimensions *Activity* and *Resource* are used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimension *Type*.
  3. The dimensions *Activity* and *Type* are used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimension *Resource*.
  4. The dimensions *Resource* and *Type* are used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimension *Activity*.
- **1-D nodes + 2-D edges**
  5. The dimension *Activity* is used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimensions *Resource* and *Type*.
  6. The dimension *Resource* is used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimensions *Activity* and *Type*.
  7. The dimension *Type* is used to define the model's event occurrences. In this case, the dependency relations are constrained by the dimensions *Activity* and *Resource*.
- **0-D nodes + 3-D edges**

8. The three dimensions (*Activity*, *Resource*, and *Type*) are used to define the model's workflow constraints. In this case, since there are no event constraints, the model will be formed by a single event occurrence representing all process events of the sub-process.

Let us consider, in this example, the process of building a multidimensional dependency graph of the model with 2-D nodes defined by dimensions *Activity* and *Resource*, and 1-D edges defined by dimension *Type*. Rather than explaining how a dependency graph can be built (process explained in Subsection 3.3.1), this example focuses on the constraining process that turns a one-dimensional dependency graph into a multidimensional one. Based on the information in the process data, $\{Activity{:}B, Resource{:}Y\}$, $\{Activity{:}C, Resource{:}X\}$, $\{Activity{:}C, Resource{:}Y\}$, $\{Activity{:}D, Resource{:}X\}$, $\{Activity{:}D, Resource{:}Y\}$, and $\{Activity{:}E, Resource{:}Z\}$ are the event constraints, while $\{Type{:}I\}$ and $\{Type{:}II\}$ are the workflow constraints.

Table 3.20 presents the counting of direct successors (i.e., $x >_W y$ in Definition 3.15), the basic information needed to build the multidimensional dependency graph. Remark that this table combines information from both event and workflow constraints. Hence, instead of simply considering six event occurrences identified above, this table describes the counting of direct successors for all the combinations of event and workflow constraints. This means that the information about the necessary measurements for building the dependency graph can be maintained in a similar structure as in the Flexible Heuristics Miner. Table 3.21 describes the combined event occurrences (event and workflow constraints) used in Table 3.20.

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}	E_{11}	E_{12}
E_1	350	0	550	0	0	0	0	0	0	0	0	0
E_2	0	150	0	0	0	450	0	0	0	0	0	0
E_3	0	0	0	0	0	0	725	0	0	0	0	0
E_4	0	0	0	0	0	0	0	450	0	0	0	0
E_5	0	0	0	0	0	0	0	0	550	0	0	0
E_6	0	0	0	0	0	0	0	0	0	800	0	0
E_7	0	0	175	0	550	0	0	0	0	0	0	0
E_8	0	0	0	0	0	0	0	0	0	0	0	450
E_9	0	0	0	0	0	0	0	0	0	0	550	0
E_{10}	0	0	0	450	0	350	0	0	0	0	0	0
E_{11}	0	0	0	0	0	0	0	0	0	0	0	0
E_{12}	0	0	0	0	0	0	0	0	0	0	0	0

Table 3.20: Direct successor counting ($x >_W y$-counting for event occurrences).

The value 550 in position (E_1,E_3) of Table 3.20 indicates that the activity B performed by resource Y (i.e., the event constraint $\{Activity{:}B, Resource{:}Y\}$) was followed 550 times (in the process data) by the activity C performed by resource X (i.e., $\{Activity{:}C, Resource{:}X\}$). The transition from the first event occurrence to the other one is characterized by the product type I (i.e., the workflow constraint $\{Type{:}I \rightarrow I\}$). The information about the counting of length-two loops (i.e., $x \gg_W y$ in Definition 3.15) can be structured – and read – in the same way as for the direct successor counting. Consequently, the same principle applies to the $x \Rightarrow_W y$-values.

Assuming the thresholds default values (i.e., $\sigma_D = \sigma_{L1L} = \sigma_{L2L} = 0.9$ and $\sigma_R = 0.05$), each step of the process of building the multidimensional dependency graph (i.e., each line of Algorithm 4) can be described as follows.

1. The first step of the algorithm is the construction of the set E, the set of all the

Event Occurrence	*Dimension Values*
E_1	$\{Activity{:}B, Resource{:}Y, Type{:}I\}$
E_2	$\{Activity{:}B, Resource{:}Y, Type{:}II\}$
E_3	$\{Activity{:}C, Resource{:}X, Type{:}I\}$
E_4	$\{Activity{:}C, Resource{:}X, Type{:}II\}$
E_5	$\{Activity{:}C, Resource{:}Y, Type{:}I\}$
E_6	$\{Activity{:}C, Resource{:}Y, Type{:}II\}$
E_7	$\{Activity{:}D, Resource{:}Y, Type{:}I\}$
E_8	$\{Activity{:}D, Resource{:}Y, Type{:}II\}$
E_9	$\{Activity{:}D, Resource{:}X, Type{:}I\}$
E_{10}	$\{Activity{:}D, Resource{:}X, Type{:}II\}$
E_{11}	$\{Activity{:}E, Resource{:}Z, Type{:}I\}$
E_{12}	$\{Activity{:}E, Resource{:}Z, Type{:}II\}$

Table 3.21: The list of event occurrences after combining the event and workflow constraints.

combinations of event and workflow constraints appearing in the process data. Remark that, each element of E is considered as an intermediate event occurrence, which is used as a stepping stone to discover the actual event occurrences that appear in the dependency graph. For the running example, all the elements of E are described in Table 3.21.

2-11. Analogue to the one-dimensional approach (Flexible Heuristics Miner). For the running example, the results of each step are the following.

- $D_1 = \{(E_1, E_1), (E_2, E_2)\}$ (step 2),
- $D_2 = \{(E_3, E_7), (E_7, E_3), (E_6, E_{10}), (E_{10}, E_6)\}$ (step 3),
- $D_{out} = \{(E_1, E_3), (E_2, E_6), ..., (E_9, E_{11}), (E_{10}, E_4)\}$ (step 4),
- $D_{in} = \{(E_1, E_3), (E_{10}, E_4), ..., (E_9, E_{11}), (E_8, E_{12})\}$ (step 5),
- $D'_{out} = \emptyset$ and $D'_{in} = \emptyset$ (steps 6 and 8), and
- $D''_{out} = D_{out}$ and $D''_{in} = D_{in}$ (i.e., no extra connections are found in steps 10 and 11).

12. Finally, like in the one-dimensional approach, the information in the different sets is combined to perform the last step of the algorithm. However, in this multi-dimensional approach, the combined information still need to be transformed to describe explicitly the dependency relations and the corresponding constraints. Therefore, the pair of – intermediate – event occurrences (a, b) should be translated into a triple consisting of two event constraints and one workflow constraint (a', b', c), with a' and b' being the event constraints of a and b, and c being the workflow constraint for the transition (a, b). In Algorithm 4, this translation is performed by the function *split*. For the running example, the translation of the pair (E_1, E_3) into (E_a, E_b, C_1) can be explained as follows.

- $D^E = \{Activity, Resource\}$ is the set of the dimensions in event constraints,
- $D^W = \{Type\}$ is the set of the dimensions in workflow constraints,
- $x = \{\ Activity{:}B, Resource{:}Y\ \} = E_a$ is the event constraint of E_1,
- $x' = \{\ Type{:}I\ \}$ is the workflow constraint of E_1,

- $y = \{\ Activity{:}C, Resource{:}X\ \} = E_b$ is the event constraint of E_3,
- $y' = \{\ Type{:}I\ \}$ is the workflow constraint of E_3, and
- $z = \{\ Type{:}I \rightarrow I\ \} = C_1$ is the workflow constraint of the transition (E_1, E_3).

Table 3.22 describes the resulting multidimensional dependency graph. Remark that the inputs of the event occurrence E_a and the outputs of the event occurrence E_f are not known due to the lack of information in the example.

Inputs	*Event Occurrence*	*Outputs*
?	E_a	$\{(E_a, C_1), (E_a, C_2), (E_b, C_1), (E_c, C_2)\}$
$\{(E_a, C_1), (E_d, C_2), (E_e, C_1)\}$	E_b	$\{(E_e, C_1), (E_e, C_2)\}$
$\{(E_a, C_2), (E_d, C_2), (E_e, C_1)\}$	E_c	$\{(E_d, C_1), (E_d, C_2)\}$
$\{(E_c, C_1), (E_c, C_2)\}$	E_d	$\{(E_b, C_2), (E_c, C_2), (E_f, C_1)\}$
$\{(E_b, C_1), (E_b, C_2)\}$	E_e	$\{(E_b, C_1), (E_c, C_1), (E_f, C_2)\}$
$\{(E_d, C_1), (E_e, C_2)\}$	E_f	?

(a) The dependency graph.

Event Occurrence	*Dimension Values*
E_a	$\{Activity{:}B, Resource{:}Y\}$
E_b	$\{Activity{:}C, Resource{:}X\}$
E_c	$\{Activity{:}C, Resource{:}Y\}$
E_d	$\{Activity{:}D, Resource{:}X\}$
E_e	$\{Activity{:}D, Resource{:}Y\}$
E_f	$\{Activity{:}E, Resource{:}Z\}$

(b) Characterization of the event occurrences in the dependency graph.

Workflow Constraint	*Dimension Values*
C_1	$\{Type{:}I \rightarrow I\}$
C_2	$\{Type{:}II \rightarrow II\}$

(c) Characterization of the workflow constraints in the dependency graph.

Table 3.22: The resulting multidimensional dependency graph.

Mining of the Event Occurrence Bindings

The second step of the Multidimensional Heuristics Miner is the identification of event occurrence bindings (i.e., pairs of input and output bindings activated together) in the multidimensional dependency graph. For each event occurrence in the dependency graph, all the possible event occurrence bindings are derived from parsing the process data on the dependency graph. Applying the mC-net definition (cf. Definition 3.10), each event occurrence is characterized by a multiset of event occurrence bindings and corresponding frequencies. Since an event occurrence binding basically consists of a pair of input and output bindings, the mining of event occurrence bindings follows a similar approach as the mining of splits and joins in the Flexible Heuristics Miner.

The process of mining the event occurrences of a multidimensional dependency graph is defined in Algorithm 5. The dependency graph of Table 3.22 is used as an illustrative example to explain this process. Let us consider $\langle \varepsilon_1 \varepsilon_2 \varepsilon_2 \varepsilon_3 \varepsilon_6 \varepsilon_4 \varepsilon_5 \varepsilon_7 \varepsilon_8 \rangle$ as the process instance being parsed. The details of the process events in the process instance are presented in Table 3.23.

Algorithm 5: Computing the Event Occurrence Bindings

Input : The process dataset W, the set of dimensions by which event occurrences are characterized (D^E), the set of dimensions by which dependency relations are characterized (D^W), and the multidimensional dependency graph DG.

Output: The multiset of event occurrence bindings and corresponding frequencies.

Method

1 $bindings \leftarrow \emptyset$; `// the multiset of event occurrence bindings`
2 **foreach** *process instance* $\delta \in W$ **do**
3 **for** $i \leftarrow 1$ **to** *length of* δ **do** $B_i^O \leftarrow \emptyset$, $B_i^I \leftarrow \emptyset$;
4 **for** $i \leftarrow 1$ **to** *length of* δ **do**
5 $e_i \leftarrow$ the process event in the i^{th} position of δ;
6 $X_i \leftarrow \{D{:}d \mid D \in D^E \ \wedge \ d$ is the value of dimension D in $e_i\}$; `// the event constraint of` e_i
7 $Y_i \leftarrow \{D{:}d \mid D \in D^W \ \wedge \ d$ is the value of dimension D in $e_i\}$; `// the workflow constraint of` e_i
8 $O_i \leftarrow$ the set of outputs of X_i in DG ; `// searching activated outputs of` X_i
9 $visitedEvents \leftarrow \emptyset$;
10 **for** $j \leftarrow (i+1)$ **to** *length of* δ **do**
11 $e_j \leftarrow$ the process event in the j^{th} position of δ;
12 $X_j \leftarrow \{D{:}d \mid D \in D^E \ \wedge \ d$ is the value of dimension D in $e_j\}$;
13 $Y_j \leftarrow \{D{:}d \mid D \in D^W \ \wedge \ d$ is the value of dimension D in $e_j\}$;
14 $Z_{i\rightarrow j} \leftarrow$`transition(`$Y_i$,$Y_j$`)`;
15 $I_j \leftarrow$ the set of inputs of X_j in DG;
16 **if** $(X_j, Z_{i\rightarrow j}) \in O_i \ \wedge \ \nexists_{(x,y)\in visitedEvents}[(x$,`transition(`$y$,$Y_j$`)`$) \in I_j]$ **then** $B_i^O \leftarrow B_i^O \cup \{(X_j, Z_{i\rightarrow j})\}$, $B_j^I \leftarrow B_j^I \cup \{(X_i, Z_{i\rightarrow j})\}$;
17 **if** $X_j = X_i \wedge Y_j = Y_i$ **then** $j \leftarrow$ length of δ **else** $visitedEvents \leftarrow visitedEvents \cup \{(X_j, Y_j)\}$;
end
18 $I_i \leftarrow$ the set of inputs of X_i in DG ; `// searching activated inputs of` X_i
19 $visitedEvents \leftarrow \emptyset$;
20 **for** $j \leftarrow (i-1)$ **to** 1 **do**
21 $e_j \leftarrow$ the process event in the j^{th} position of δ;
22 $X_j \leftarrow \{D{:}d \mid D \in D^E \ \wedge \ d$ is the value of dimension D in $e_j\}$;
23 $Y_j \leftarrow \{D{:}d \mid D \in D^W \ \wedge \ d$ is the value of dimension D in $e_j\}$;
24 $Z_{j\rightarrow i} \leftarrow$`transition(`$Y_j$,$Y_i$`)`;
25 $O_j \leftarrow$ the set of outputs of X_j in DG;
26 **if** $(X_j, Z_{j\rightarrow i}) \in I_i \ \wedge \ \nexists_{(x,y)\in visitedEvents}[(x$,`transition(`$Y_j$,$y$`)`$) \in O_j]$ **then** $B_i^I \leftarrow B_i^I \cup \{(X_j, Z_{j\rightarrow i})\}$, $B_j^O \leftarrow B_j^O \cup \{(X_i, Z_{j\rightarrow i})\}$;
27 **if** $X_j = X_i \wedge Y_j = Y_i$ **then** $j \leftarrow 1$ **else** $visitedEvents \leftarrow visitedEvents \cup \{(X_j, Y_j)\}$;
end
end
28 **for** $i \leftarrow 1$ **to** *length of* δ **do**
29 $e_i \leftarrow$ the process event in the i^{th} position of δ;
30 $X_i \leftarrow \{D{:}d \mid D \in D^E \ \wedge \ d$ is the value of dimension D in $e_i\}$;
31 $bindings \leftarrow bindings \cup (B_i^I, B_i^O)^{X_i}$;
end
end
32 **return** *bindings*;

function transition(C_1,C_2) $= \{D{:}d_1 \rightarrow d_2 \mid D{:}d_1 \in C_1 \wedge D{:}d_2 \in C_2\}$

Process Event	*Activity*	*Resource*	*Type*	...
ε_1	A	Z	I	
ε_2	B	Y	I	
ε_3	C	X	I	
ε_4	C	Y	I	
ε_5	D	X	I	
ε_6	D	Y	I	
ε_7	E	Z	I	
ε_8	F	Z	I	
...				

Table 3.23: The characterization of process events according to the attributes in the event log.

Every event occurrence binding is derived from parsing a process instance on the dependency graph. Each event of a process instance is characterized by an input and an output binding (i.e., a join and a split). This pair of – input and output – bindings define the event occurrence binding of the event. The set of all event occurrence bindings derived from parsing the process data on the dependency graph defines the mC-net.

The mining of the event occurrence binding of the process event ε_3 can be described as follows. Note that ε_3 appears in the fourth position of the process instance (i.e., $i = 4$ in line 4 and e_4 in line 5). Considering that the dimensions *Activity* and *Resource* are used to define the model's event occurrences, the event constraint of ε_3 is $X_4 = \{Activity{:}C, Resource{:}X\}$ (line 6). Analogously, since the dimension *Type* is used to characterize the model's dependency relations, the workflow constraint of ε_3 is $Y_4 = \{Type{:}I\}$ (line 7). According to Table 3.22, the outputs of X_4 are $(\{Activity{:}D, Resource{:}Y\}, \{Type{:}I \rightarrow I\})$ and $(\{Activity{:}D, Resource{:}Y\}, \{Type{:}II \rightarrow II\})$ (i.e., set O_4 in line 8). The output binding of ε_3 is defined by the subsequence of process events from the instance's fifth position onwards (i.e., $j = 5$ to 9 in the loop of line 10). A pair of – event and workflow – constraints (X_k, Y_k) that characterizes a process event e_k in this subsequence is part of the output binding of ε_3 if (line 16):

- Being $Z_{4\rightarrow k} = transition(Y_4, Y_k)$ the workflow constraint of the transition (e_4, e_k), $(X_k, Z_{4\rightarrow k})$ is part of the set of outputs of ε_3,[10] and
- There is no other pair of constraints (X_l, Y_l) in between of ε_3 (e_4) and e_k (i.e., set *visitedEvents* in lines 9, 16 and 17) such that $(X_l, transition(Y_k, Y_l))$ is part of the set of inputs of e_k. This condition checks if this possible output is actually activated by ε_3 and not by any other process event.

The pair (X_5, Y_5) (ε_6 in line 11, $X_5 = \{Activity{:}D, Resource{:}Y\}$ in line 12, and $Y_5 = \{Type{:}I\}$ in line 13) satisfies these conditions. $Z_{4\rightarrow 5} = \{Type{:}I \rightarrow I\}$ is the workflow constraint of the transition $(\varepsilon_3, \varepsilon_6)$ (line 14). $(X_5, Z_{4\rightarrow 5}) \in O_4$ and there is not any process event in between of ε_3 and ε_6. Therefore, the output binding of ε_3 in the given process instance is defined as an XOR-Split $\{(\{Activity{:}D, Resource{:}Y\}, \{Type{:}I \rightarrow I\})\}$.

[10] The function $transition(C_1, C_2)$ converts two constraints composed by simple values into a constraint composed by transition values. For example, $transition(\{D_1{:}x, D_2{:}y\}, \{D_1{:}x, D_2{:}z\}) = \{D_1{:}x \rightarrow x, D_2{:}y \rightarrow z\}$.

The mining of input bindings (lines 18-27) is analogous to the mining of output bindings (lines 8-17). The input binding of ε_3 in the given process instance is defined as an XOR-Join $\{(\{Activity{:}B, Resource{:}Y\}, \{Type{:}I \rightarrow I\})\}$. Finally, the event occurrence binding of ε_3 is defined as the pair with the input and output bindings (line 31):

$$(\{(\{Activity{:}B, Resource{:}Y\}, \{Type{:}I \rightarrow I\})\},\{(\{Activity{:}D, Resource{:}Y\}, \{Type{:}I \rightarrow I\})\})$$

Table 3.24 summarizes the possible input and output bindings in the dependency graph of Table 3.22. Similarly, Table 3.25 describes the possible event occurrence bindings in the dependency graph. The results in these tables are based on the event log from which the model of Figure 3.17 is generated.

Input Bindings	*Event Occurrence*	*Output Bindings*
$[\ \{(?, C_1)\}^{550}, \{(?, C_2)\}^{450}\ ]$	E_a	$[\ \{(E_a, C_1)\}^{350}, \{(E_a, C_2)\}^{150}, \{(E_b, C_1)\}^{550}, \{(E_c, C_2)\}^{450}\ ]$
$[\ \{(E_a, C_1)\}^{550}, \{(E_d, C_2)\}^{450}, \{(E_e, C_1)\}^{175}\ ]$	E_b	$[\ \{(E_e, C_1)\}^{725}, \{(E_e, C_2)\}^{450}\ ]$
$[\ \{(E_a, C_2)\}^{450}, \{(E_d, C_2)\}^{350}, \{(E_e, C_1)\}^{550}\ ]$	E_c	$[\ \{(E_d, C_1)\}^{550}, \{(E_d, C_2)\}^{800}\ ]$
$[\ \{(E_c, C_1)\}^{550}, \{(E_c, C_2)\}^{800}\ ]$	E_d	$[\ \{(E_b, C_2)\}^{450}, \{(E_c, C_2)\}^{350}, \{(E_f, C_1)\}^{550}\ ]$
$[\ \{(E_b, C_1)\}^{725}, \{(E_b, C_2)\}^{450}\ ]$	E_e	$[\ \{(E_b, C_1)\}^{175}, \{(E_c, C_1)\}^{550}, \{(E_f, C_2)\}^{450}\ ]$
$[\ \{(E_d, C_1)\}^{550}, \{(E_e, C_2)\}^{450}\ ]$	E_f	$[\ \{(?, C_1)\}^{540}, \{(?, C_2)\}^{447}, \emptyset^{13}\ ]$

Table 3.24: All input and output bindings – and corresponding frequencies – of the multidimensional dependency graph of Table 3.22.

Figure 3.18 depicts the mC-net based on the event occurrence bindings of Table 3.25. Note that, following the mC-net definition, the numerical information in nodes represent the activity frequency. Analogously, that information on edges represent the number of times a transition from an activity to another occurred.

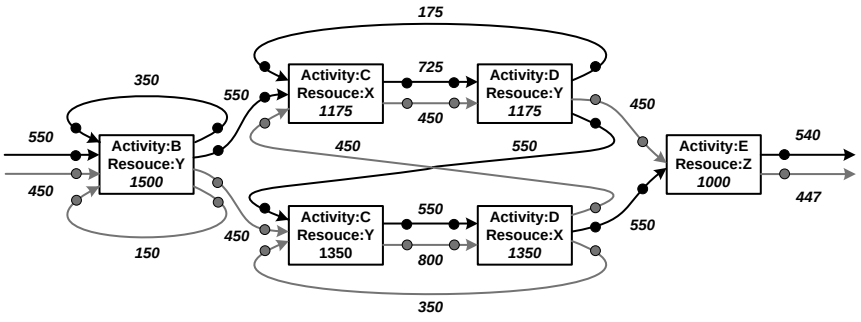


Figure 3.18: The resulting mC-net.

Event Occurrence	*Event Occurrence Bindings*
E_a	$[\ (\{(?,C_1)\},\{(E_a,C_1)\})^{350}, (\{(?,C_2)\},\{(E_a,C_2)\})^{150},$ $(\{(?,C_1)\},\{(E_b,C_1)\})^{200}, (\{(?,C_2)\},\{(E_c,C_2)\})^{300},$ $(\{(E_a,C_1)\},\{(E_b,C_1)\})^{350}, (\{(E_a,C_2)\},\{(E_c,C_2)\})^{150}\ ]$
E_b	$[\ (\{(E_a,C_1)\},\{(E_e,C_1)\})^{550}, (\{(E_d,C_2)\},\{(E_e,C_2)\})^{450},$ $(\{(E_e,C_1)\},\{(E_e,C_1)\})^{175}\ ]$
E_c	$[\ (\{(E_a,C_2)\},\{(E_d,C_2)\})^{450}, (\{(E_d,C_2)\},\{(E_d,C_2)\})^{350},$ $(\{(E_e,C_1)\},\{(E_d,C_1)\})^{550}\ ]$
E_d	$[\ (\{(E_c,C_1)\},\{(E_f,C_1)\})^{550}, (\{(E_c,C_2)\},\{(E_c,C_2)\})^{350},$ $(\{(E_c,C_2)\},\{(E_b,C_2)\})^{450}\ ]$
E_e	$[\ (\{(E_b,C_1)\},\{(E_c,C_1)\})^{550}, (\{(E_b,C_1)\},\{(E_b,C_1)\})^{175},$ $(\{(E_b,C_2)\},\{(E_f,C_2)\})^{450}\ ]$
E_f	$[\ (\{(E_d,C_1)\},\{(?,C_1)\})^{540}, (\{(E_e,C_2)\},\{(?,C_2)\})^{447},$ $(\{(E_d,C_1)\},\emptyset)^{10}, (\{(E_e,C_2)\},\emptyset)^{3}\ ]$

Table 3.25: All event occurrence bindings – and corresponding frequencies – of the multidimensional dependency graph of Table 3.22.

3.4 Summary

Five components of the multidimensional process discovery approach were discussed in this chapter.

- **Process data** are characterized in Section 3.1 in terms of structure.
- **Augmented Causal nets** are presented in Subsection 3.2.1 as instances of traditional process models. Augmented Causal nets are characterized by their simplicity and flexibility to describe complex behavior.
- **Multidimensional Causal nets** are introduced in Subsection 3.2.2 as instances of multidimensional process models. A multidimensional Causal net can be seen as a generalization of an augmented Causal net. Instead of grouping process events simply by activity, process events can be grouped by multiple aspects of the business process. Furthermore, constructs can also be characterized by multiple aspects.
- **Flexible Heuristics Miner** is presented in Subsection 3.3.1 as a – traditional – process discovery technique focusing on the control-flow perspective. Based on the Heuristics Miner [131, 132], this technique can be used to construct an augmented Causal net from process data. A great advantage of this technique is its capability of dealing with noise and infrequent (or incomplete) behavior.
- **Multidimensional Heuristics Miner** is described in Subsection 3.3.2 as a multidimensional process discovery technique focusing on the control-flow perspective. The Multidimensional Heuristics Miner can be seen as a generalization of the Flexible Heuristics Miner. Instead of one-dimensional process models (over the dimension *Activity*), this technique can build process models over multiple dimensions.

Except the process data component, all components described above are implemented as ProM 6 plugins, or objects that can be executed in the ProM framework [123, 125].

Answering research questions q_1 and q_2, the link between traditional and multi-dimensional process discovery is established by the generalizations of (i) augmented Causal nets to multidimensional Causal nets, and (ii) the Flexible Heuristics Miner to the Multidimensional Heuristics Miner.

Chapter 4

Process Similarity

In this chapter, we introduce a methodology to compute differences or similarities in multidimensional process models. Taking a multidimensional process model as reference, it is intended to define similarity functions for comparing:

- A pair of **event occurrences** in multidimensional process model.
- A pair of **event occurrence bindings** in multidimensional process model.
- A pair of **sequences of event occurrence bindings** describing process instances according to the multidimensional process model.
- A pair of **multisets of event occurrence bindings** describing sub-processes in the multidimensional process model.

This kind of information can be used to identify similarities in the process behavior of multidimensional process models. In this thesis, process similarity is used in the extraction of non-trivial patterns from multidimensional process models (cf. Section 6.2). Process retrieval [139, 140] is another possible application.

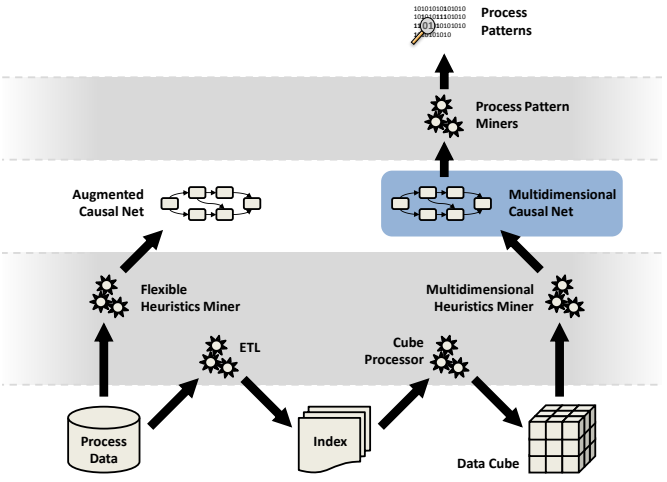


Figure 4.1: Positioning of Chapter 4 with respect to the multidimensional process discovery approach.

Figure 4.1 positions this chapter with respect to the multidimensional process discovery approach. In the next sections, we firstly introduce the concept of similarity in multidimensional process models. Then, similarity functions are introduced to determine similarity between (i) event occurrences, (ii) event occurrence bindings, (iii) process instances, and (iv) sub-processes in a multidimensional process model.

4.1 Process Similarity in Traditional Process Models

Process similarity can be defined as a measure that quantifies how identical two process models are. The main approaches for assessing process similarity between – traditional – process models can be described as follows.

- **Similarity based on process elements** takes into account nodes and edges in the process models (but not the control-flow). In [35], several similarity measures based on node aspects are studied. Four different aspects can be considered in the similarity assessment of process elements:
  - The *syntactic* aspect compares the designations (labels) of two nodes (or two node attributes) according to their string-edit distance.[1]
  - The *semantic* aspect compares the designations (labels) of two nodes (or two node attributes) according to their meanings.
  - The *attribute* aspect compares the attribute values of two nodes.
  - The *type* aspect compares the types of two nodes.

  All these aspects are quantified in a similarity score between 0 (no similarity) and 1 (identical elements). In [85], it is proposed a similarity measure based on the percentage of activities (nodes) and dependency relations (edges) that appear in both process models. This similarity measure relies on exact node matchings. In [56], a similar approach supporting non-exact node matchings (like the similarity aspects described above) is proposed.
- **Similarity based on process structure** takes into account the edit distance between graphs. In [35], a similarity measure based on graph-edit operations is discussed. *Node deletion or insertion*, *node substitution*, and *edge deletion or insertion* are the graph-edit operations considered in this approach. Eventually, other graph-edit operations may also be considered [45]. Furthermore, this approach may be combined with other similarity metrics (e.g., combining string- and graph-edit distances).
- **Similarity based on process behavior** takes into account the dependency graphs (including the splits and joins constructs) of the process models. In [129], the process behavior of two process models are compared by analyzing the causal dependencies in their dependency graphs. If two causal dependencies share specific characteristics then the process behavior represented by the dependencies is considered similar. In [35], a similarity measure based on *causal footprints* (i.e., the input and output relationships of activities) is introduced. In this approach

[1] The edit distance between two objects quantifies the number of atomic operations necessary to transform one object into the other. For instance, the Levenshtein distance, a string-edit distance metric, consists of the minimum number of insertion, deletion, and substitution operations (on single characters) to transform one string into the other.

the behavioral similarity of two process models is determined by computing their distance in a vector space constructed from their causal footprints.

- **Similarity based on process instances** takes into account the sequences of process events described in the models. These sequences of process events can be either generated from a process model by inference or simulation, or obtained from actual process executions. In [42], it is presented a similarity measure based on the longest common subsequence of process events in process instances inferred from the models in comparison. In [5], frequent process instances from actual process executions are used to calculate the similarity of process models.

For further details about business process similarity measures see [16, 35].

4.2 Process Similarity in Multidimensional Process Models

In this section, we present an approach to calculate similarity in multidimensional process models. Rather than comparing two process models describing different process perspectives, we exclusively focus on the **comparison of sub-processes** in a process model. This means that, since the process elements are characterized by the same dimensions, the comparison of sub-processes does not rely on any of the *syntactic*, *semantic*, *attribute*, and *type* aspects identified in the last section, but on the direct comparison of the dimension values that describe the process elements.

In multidimensional process models, nodes represent event occurrences, which are characterized by a set of dimension values. Since attribute values can be explicitly represented in nodes, it is not necessary to take the attribute aspect into account for assessing the similarity between event occurrences. Additionally, since there is the assumption that dimensions and dimension values are unambiguous, the similarity between nodes can rely on exact matchings of dimension values (instead of non-exact matchings based on syntax and semantics). Finally, since nodes in multidimensional process models are all of the same type, the type aspect is also not relevant to event occurrence similarity. In Subsection 4.2.1, we define a similarity metric to compare event occurrences. Basically, this metric consists of the weighted average of matched dimension values.

In multidimensional process models, edges represent dependency relations between event occurrences. Like event occurrences, dependency relations are also characterized by a set of dimension values. Therefore, computing similarity between edges of a multidimensional process model can follow a similar strategy as in computing similarity between nodes. A similarity metric to compare dependency relations is defined in Subsection 4.2.2.

Rather than structure, we are interested in assessing the process similarity by comparing process behavior. To do so, we introduce a number of behavioral-based similarity metrics to compare elements of a multidimensional process model as well as sub-processes of it. Note that these metrics are based on similarity metrics for traditional models described in [16]. The mC-net as depicted in Figure 4.2 is used as running example to illustrate the similarity computation in a multidimensional process model. In this example, it is intended to explain iteratively how the similarity between the sub-processes α and β in the mC-net can be computed. The event occurrence bindings

of every event occurrence of the mC-net are provided in Table 4.1. Details about this calculation process are provided in the next subsections.

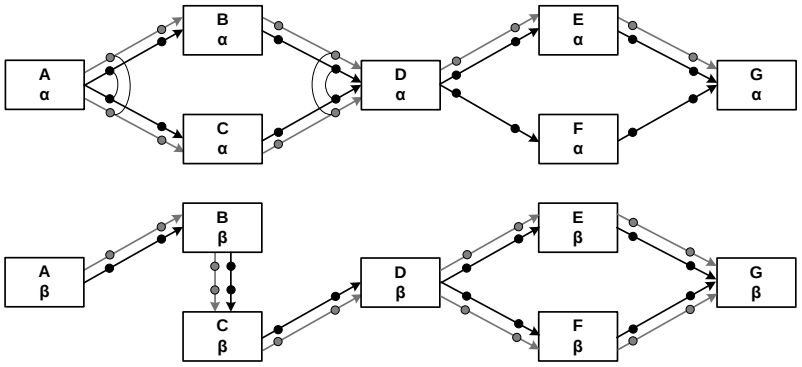


Figure 4.2: A three-dimensional process model.

4.2.1 Similarity between Event Occurrences

Comparing event occurrences basically consists of matching the dimension values that characterize the different occurrences. The values that cannot be matched define the differences between the event occurrences. Remark that a matching focuses always on values from the same dimension. For instance, the event occurrences $E = \{D_1{:}X, D_2{:}Y\}$ and $E' = \{D_1{:}Y, D_2{:}X\}$ (D_1 and D_2 are dimensions sharing the same domain: $\{X, Y\}$) do not have any dimension value in common once that $X \neq Y$ for D_1, and $Y \neq X$ for D_2.

By counting the matched values, it is possible to quantify the similarity between event occurrences. Basically, this measure consists of the weighted ratio of matched values to the number of dimensions.

Definition 4.1 (Event Occurrences Similarity) *Let an event occurrence X of an n-dimensional process model be defined as a set of dimension values $\{D_1{:}x_1, ..., D_n{:}x_n\}$. The similarity of two event occurrences A and B (from the same model) can be defined as follows.*

$$corr(A,B) = \begin{cases} 1 & \text{if } n = 0 \\ \frac{\sum_{D:a \in A, D:b \in B} equals(D{:}a, D{:}b)}{\sum_{D:a \in A} card(D)} & \text{otherwise} \end{cases} \tag{4.1}$$

$$equals(D:x, D:y) = \begin{cases} card(D) & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Remark that $\text{card}(D) \in \mathbb{N}$ *represents the cardinality of the dimension D (i.e., the number of dimension values in D). In this approach there is the assumption that two distinct dimension values have no similarity relation to each other.*

For the similarity assessment, dimensions with many values (i.e., higher cardinality) are considered to be more relevant than low-cardinality dimensions. The rationale behind this consideration can be demonstrated by the following example. Taking the

ID	Inputs		Outputs	
			B_α	C_α
a_1		$\mapsto$	●	●
a_2		$\mapsto$	●	●

(a) Bindings of A_α.

ID	Inputs		Outputs
			B_β
a_3		$\mapsto$	●
a_4		$\mapsto$	●

(b) Bindings of A_β.

ID	Inputs		Outputs
	A_α		D_α
b_1	●	$\mapsto$	●
b_2	●	$\mapsto$	●

(c) Bindings of B_α.

ID	Inputs		Outputs
	A_β		C_β
b_3	●	$\mapsto$	●
b_4	●	$\mapsto$	●

(d) Bindings of B_β.

ID	Inputs		Outputs
	A_α		D_α
c_1	●	$\mapsto$	●
c_2	●	$\mapsto$	●

(e) Bindings of C_α.

ID	Inputs		Outputs
	B_β		D_β
c_3	●	$\mapsto$	●
c_4	●	$\mapsto$	●

(f) Bindings of C_β.

ID	Inputs			Outputs	
	B_α	C_α		E_α	F_α
d_1	●	●	$\mapsto$	●	
d_2	●	●	$\mapsto$	●	
d_3	●	●	$\mapsto$		●

(g) Bindings of D_α.

ID	Inputs			Outputs	
	C_β			E_β	F_β
d_4	●		$\mapsto$	●	
d_5	●		$\mapsto$	●	
d_6	●		$\mapsto$		●
d_7	●		$\mapsto$		●

(h) Bindings of D_β.

ID	Inputs		Outputs
	D_α		G_α
e_1	●	$\mapsto$	●
e_2	●	$\mapsto$	●

(i) Bindings of E_α.

ID	Inputs		Outputs
	D_β		G_β
e_3	●	$\mapsto$	●
e_4	●	$\mapsto$	●

(j) Bindings of E_β.

ID	Inputs		Outputs
	D_α		G_α
f_1	●	$\mapsto$	●

(k) Bindings of F_α.

ID	Inputs		Outputs
	D_β		G_β
f_2	●	$\mapsto$	●
f_3	●	$\mapsto$	●

(l) Bindings of F_β.

ID	Inputs			Outputs
	E_α	F_α		
g_1	●		$\mapsto$	
g_2	●		$\mapsto$	
g_3		●	$\mapsto$	

(m) Bindings of G_α.

ID	Inputs			Outputs
	E_β	F_β		
g_4	●		$\mapsto$	
g_5	●		$\mapsto$	
g_6		●	$\mapsto$	
g_7		●	$\mapsto$	

(n) Bindings of G_β.

Table 4.1: The event occurrence bindings of Figure 4.2's mC-Net.

multidimensional process model in Figure 4.2 into account, let us calculate the event occurrences similarity between $\binom{B}{\alpha} = \{D_1{:}B, D_2{:}\alpha\}$ and $\binom{B}{\beta} = \{D_1{:}B, D_2{:}\beta\}$. Applying the definition provided above, the similarity value is:

$$\begin{aligned} corr\left(\begin{matrix}B & B\\ \alpha & \beta\end{matrix}\right) &= corr(\{D_1 : B, D_2 : \alpha\}, \{D_1 : B, D_2 : \beta\}) \\ &= \frac{equals(D_1 : B, D_1 : B) + equals(D_2 : \alpha, D_2 : \beta)}{card(D_1) + card(D_2)} \\ &= \frac{7+0}{7+2} = 0.78, \end{aligned}$$

where $card(D_1) = |\{A, B, C, D, E, F, G\}| = 7$ and $card(D_2) = |\{1, 2\}| = 2$. Similarly, the similarity value for $\binom{B}{\alpha} = \{D_1{:}B, D_2{:}\alpha\}$ and $\binom{C}{\alpha} = \{D_1{:}C, D_2{:}\alpha\}$ is:

$$corr\left(\begin{matrix}B & C\\ \alpha & \alpha\end{matrix}\right) = \frac{0+2}{7+2} = 0.22.$$

In both cases, one out of two dimension values of $\binom{B}{\alpha}$ has a match in $\binom{B}{\beta}$ and $\binom{C}{\alpha}$. Without considering the dimensions cardinality as a weight (i.e., the cardinality values are replaced by 1), both cases would have the same similarity value ($corr\binom{B}{\alpha}{\beta} = corr\binom{B}{\alpha}{\alpha} = 0.5$). This would mean that D_1 and D_2 were treated equally even though the likelihoods of their values appearing in an event occurrence are uneven. The dimensions cardinality are used to overcome this issue. The idea is to adjust the similarity value by giving more importance to the high-cardinality dimensions in order to prioritize the less likely matchings. Therefore, since the cardinality of D_1 is higher than the cardinality of D_2, $corr\binom{B}{\alpha}{\beta}$ is greater than $corr\binom{B}{\alpha}{\alpha}$.

Remark that the presented approach relies on the dimensions cardinality because this information can always be computed from the process data. If there is a function $f(D) \in \mathbb{R}$ that determines the importance (weight) of the dimension D then, by replacing $card(D)$ by $f(D)$ in Definition 4.1, a different notion of importance of dimensions may be used instead of the dimensions cardinality.

4.2.2 Similarity between Event Occurrence Bindings

The comparison of event occurrence bindings focuses on three aspects: (i) the event occurrences to which the bindings refer, (ii) the bindings' activated inputs, and (iii) the bindings' activated outputs. The similarity between event occurrence bindings is defined by the combination of the similarities of these aspects.

Definition 4.2 (Event Occurrence Bindings Similarity) *Let a event occurrence binding M^X of an event occurrence X be defined as the tuple $(I, O)^X$, where $I = \{i_1, ..., i_\eta\}$ and $O = \{o_1, ..., o_\mu\}$ are the activated inputs and outputs of X. Every activated input i_x ($x \in [1, \eta]$) or output $o_{x'}$ ($x' \in [1, \mu]$) is defined by the tuple (C^E, C^W), where $C^E = \{v_1, ..., v_n\}$ and $C^W = \{v'_1, ..., v'_m\}$ are the event occurrence and workflow constraints. The similarity of two event occurrence bindings $A^X = (I_A, O_A)^X$ and $B^Y = (I_B, O_B)^Y$ can be defined as follows.*

$$sim_B(A^X, B^Y) = \frac{corr(X, Y) + sim_{IO}(I_A, I_B) + sim_{IO}(O_A, O_B)}{3}, \qquad (4.3)$$

where $\mathrm{corr}(X,Y)$ *determines the similarity of the event occurrences* X *and* Y *(cf. Definition 4.1).*

$$sim_{IO}(S,S') = \begin{cases} 1 & \text{if } S=\emptyset \ \wedge \ S'=\emptyset \\ 1/(|S|+|S'|) \times \Big(\sum_{s\in S} \max_{s'\in S'} \big(sim_{AV}(s,s')\big) & \\ \quad + \sum_{s'\in S'} \max_{s\in S} \big(sim_{AV}(s',s)\big)\Big) & \text{if } S\neq\emptyset \ \wedge \ S'\neq\emptyset \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

$$sim_{AV}\big((C_1^E,C_1^W),(C_2^E,C_2^W)\big) = \frac{corr(C_1^E,C_2^E)+corr(C_1^W,C_2^W)}{2} \tag{4.5}$$

The following example is used to explain how the similarity between event occurrence bindings is calculated. Let us consider the following event occurrence bindings from the multidimensional process model in Figure 4.2:

$$A^X = (\{\{B,\alpha\}^{\{\bullet\}},\{C,\alpha\}^{\{\bullet\}}\},\{\{E,\alpha\}^{\{\bullet\}}\})^{\{D,\alpha\}}, \text{ and}$$
$$B^Y = (\{\{C,\beta\}^{\{\bullet\}}\},\{\{E,\beta\}^{\{\bullet\}}\})^{\{D,\beta\}}.$$

Applying the definition provided above, the similarity between A^X and B^Y is:

$$\begin{aligned} sim_B(A^X,B^Y) &= \frac{corr(X,Y)+sim_{IO}(I_A,I_B)+sim_{IO}(O_A,O_B)}{3} \\ &= 1/3 \times \big(corr(\{D,\alpha\},\{D,\beta\}) \\ &\quad + sim_{IO}(\{\{B,\alpha\}^{\{\bullet\}},\{C,\alpha\}^{\{\bullet\}}\},\{\{C,\beta\}^{\{\bullet\}}\}) \\ &\quad + sim_{IO}(\{\{E,\alpha\}^{\{\bullet\}}\},\{\{E,\beta\}^{\{\bullet\}}\})\big) \\ &= 1/3 \times \big(\frac{7+0}{7+2} + \frac{(0.5+0.89)+0.89}{2+1} + \frac{0.89+0.89}{1+1}\big) \\ &= 1/3 \times (0.78+0.76+0.89) = 0.81 \end{aligned}$$

As mentioned before, the event occurrence bindings similarity consists of the average of the similarity values of (i) the event occurrences to which the bindings refer (i.e., $X=\{D,\alpha\}$ and $Y=\{D,\beta\}$), (ii) the bindings' activated inputs (i.e., $I_A=\{\{B,\alpha\}^{\{\bullet\}},\{C,\alpha\}^{\{\bullet\}}\}$ and $I_B=\{\{C,\beta\}^{\{\bullet\}}\}$), and (iii) the bindings' activated outputs (i.e., $I_A=\{\{E,\alpha\}^{\{\bullet\}}\}$ and $I_B=\{\{E,\beta\}^{\{\bullet\}}\}$). The computation of the similarity value of the event occurrences X and Y is explained in Section 4.2.1. The similarity value of the bindings' activated inputs consists of the ratio of the summation of the best inputs matching value to the total number of inputs (in both bindings). The input matching value can be defined as the average of similarity values of the inputs' constraints (event occurrences and workflow). Therefore, for instance, the inputs match value of 0.5 is obtained from:

$$\begin{aligned} sim_{AV}\big((\{B,\alpha\},\{\bullet\}),(\{C,\beta\},\{\bullet\})\big) &= \frac{corr(\{B,\alpha\},\{C,\beta\})+corr(\{\bullet\},\{\bullet\})}{2} \\ &= 1/2 \times \Big(\frac{0+0}{7+2}+\frac{2}{2}\Big) = 1/2 \times 1 = 0.5 \end{aligned}$$

Note that the function $corr(X, Y)$ can be used to determine the similarity between either event occurrences or workflow constraints. The similarity value of the bindings' activated outputs is analogous to that for activated inputs.

4.2.3 Similarity between Process Instances

The similarity between process instances is based on sequences of event occurrence bindings. Since the event occurrence bindings implicitly describe the process behavior, the process instances similarity can be achieved through the comparison of multisets of event occurrence bindings.

Definition 4.3 (Process Instances Similarity) *Let a process instance T of a multidimensional process model be a finite sequence of event occurrences $\langle E_1 E_2 ... E_n \rangle$ that can be translated into the sequence of event occurrence bindings $T' = \langle M_1^{E_1} M_2^{E_2} ... M_n^{E_n} \rangle$ (cf. Subsection 3.2.2). The similarity of two process instances T_1 and T_2 can be defined as follows.*

$$sim_{PI}(T_1, T_2) = \begin{cases} 1 & \text{if } |T_1'| = 0 \ \wedge \ |T_2'| = 0 \\ \frac{min(|T_1'|,|T_2'|)}{max(|T_1'|,|T_2'|)} \times \frac{sim_{ID}\left(id(T_1'), id(T_2')\right)}{|T_1'|+|T_2'|} & \text{if } |T_1'| \neq 0 \ \wedge \ |T_2'| \neq 0 \\ 0 & \text{otherwise} \end{cases}, \tag{4.6}$$

where $|T'|$ represents the number of elements in the sequence of event occurrence bindings T'. The function $id(T')$ is defined in Definition 6.7.

$$sim_{ID}(X, Y) = \sum_{x \in X} \max_{y \in Y} \left(sim_{BME}(x, y)\right) + \sum_{y \in Y} \max_{x \in X} \left(sim_{BME}(y, x)\right) \tag{4.7}$$

$$sim_{BME}\left((a^{\alpha}), (b^{\beta})\right) = \frac{min(\alpha, \beta)}{max(\alpha, \beta)} \times sim_B(a, b), \tag{4.8}$$

where $sim_B(a, b)$ determines the similarity of the event occurrence bindings a and b (cf. Definition 4.2).

The following example is used to explain how to calculate the similarity between process instances. Let us consider the following process instances from the multidimensional process model in Figure 4.2: $T_1 = \langle {}^{A\ C\ B\ D\ E\ G}_{\alpha\ \alpha\ \alpha\ \alpha\ \alpha\ \alpha} \rangle^{\bullet}$ and $T_2 = \langle {}^{A\ B\ C\ D\ E\ G}_{\beta\ \beta\ \beta\ \beta\ \beta\ \beta} \rangle^{\bullet}$. Note that these process instances can be converted first into the sequences of event occurrence bindings $T_1' = \langle a_2 c_2 b_2 d_2 e_2 g_2 \rangle$ and $T_2' = \langle a_4 b_4 c_4 d_5 e_4 g_5 \rangle$, and then into multisets of event occurrence bindings as described in Definition 6.7. Applying the definition provided above, the similarity between T_1 and T_2 is:

$$
\begin{aligned}
sim_{PI}(T_1, T_2) &= \frac{min(|T_1'|, |T_2'|)}{max(|T_1'|, |T_2'|)} \times \frac{sim_{ID}\big(id(T_1'), id(T_2')\big)}{|T_1'| + |T_2'|} \\
&= \frac{min(6,6)}{max(6,6)} \times \frac{1}{6+6} \times sim_{ID}([a_2, b_2, c_2, d_2, e_2, g_2], \\
&\quad [a_4, b_4, c_4, d_5, e_4, g_5]) \\
&= 6/6 \times 1/12 \times \big((0.85 + 0.72 + 0.72 + 0.81 + 0.85 + 0.89) \\
&\quad + (0.85 + 0.72 + 0.72 + 0.81 + 0.85 + 0.89)\big) \\
&= 0.81
\end{aligned}
$$

Basically, the process instances similarity consists of the ratio of the summation of the best bindings matching value to the total number of bindings (in both process instances). The bindings matching value can be defined as the bindings similarity. Both similarity and matching values are adjusted to the proportion of their lengths (process instances) and frequencies (event occurrence bindings). Therefore, for instance, the event occurrence bindings match values of 0.81 are obtained from:

$$
\begin{aligned}
sim_{BME}\big((d_2^1), (d_5^1)\big) &= \frac{min(1,1)}{max(1,1)} \times sim_B(d_2, d_5) \\
&= 1 \times sim_B\big((\{\{B, \alpha\}^{\{\bullet\}}, \{C, \alpha\}^{\{\bullet\}}\}, \{\{E, \alpha\}^{\{\bullet\}}\})^{\{D,\alpha\}}, \\
&\quad (\{\{C, \beta\}^{\{\bullet\}}\}, \{\{E, \beta\}^{\{\bullet\}}\})^{\{D,\beta\}}\big) \\
&= 1/3 \times \big(corr(\{D, \alpha\}, \{D, \beta\}) \\
&\quad + sim_{IO}(\{B, \alpha\}^{\{\bullet\}}, \{C, \alpha\}^{\{\bullet\}}\}, \{C, \beta\}^{\{\bullet\}}\}) \\
&\quad + sim_{IO}(\{\{E, \alpha\}^{\{\bullet\}}, \{\{E, \beta\}^{\{\bullet\}})\big) \\
&= \frac{1}{3} \times \Big(\frac{7+0}{7+2} + \frac{(0.5 + 0.89) + 0.89}{2+1} + \frac{0.89 + 0.89}{1+1}\Big) \\
&= 0.81 = sim_{BME}\big((d_5^1), (d_2^1)\big)
\end{aligned}
$$

4.2.4 Similarity between Sub-Processes

Like for any process instance, any sub-process of a multidimensional process model can be univocally described by a multiset of event occurrence bindings. Therefore, the similarity between sub-processes can be calculated using the same approach as for process instances. The only difference is that, instead of two sequences of event occurrence bindings, Equation 4.6 should get two multisets of event occurrence bindings as arguments.

Definition 4.4 (Process Instances Similarity) *Let a sub-process S of a multidimensional process model be a multiset of event occurrence bindings $[a_1^{\alpha_1}, ..., a_n^{\alpha_n}]$ (cf. Definition 6.7). The similarity of two sub-processes S_1 and S_2 can be defined as follows.*

$$
sim_{SP}(S_1, S_2) = \begin{cases} 1 & \text{if } |S_1| = 0 \ \wedge \ |S_2| = 0 \\ \frac{min(|S_1|,|S_2|)}{max(|S_1|,|S_2|)} \times \frac{sim_{ID}(S_1,S_2)}{|S_1|+|S_2|} & \text{if } |S_1| \neq 0 \ \wedge \ |S_2| \neq 0 \\ 0 & \text{otherwise} \end{cases}, \quad (4.9)
$$

where $|S|$ represents the number of elements in the multiset of event occurrence bindings S. The function sim_{ID} is defined in Definition 4.3.

4.3 Summary

The similarity aspect of multidimensional process models was discussed in this chapter. Similarity in multidimensional process models can be calculated by comparing event occurrences (cf. Subsection 4.2.1), event occurrence bindings (cf. Subsection 4.2.2), sequences of event occurrence bindings (cf. Subsection 4.2.3), or multisets of event occurrence bindings (cf. Subsection 4.2.4). All similarity functions presented in this chapter are implemented in an object that can be executed in ProM [123, 125].

Addressing research question q_2, the methodology presented in this chapter provides the link between traditional and multidimensional process models in terms of similarity assessment.

Chapter 5

Process Analysis

In this chapter, we introduce a framework in which the dimensionality of process data (i.e., the different attributes) can be exploited for process discovery and analysis. Combining concepts from OLAP and process mining, it is possible to organize the process data in such a way that the process analysis can be performed taking into account multiple dimensions. This process consists of

i. retrieving the process data from the sources (e.g., an event log),
ii. deriving relevant information for analysis from the data,
iii. computing summaries for the different dimensions of the business process in analysis, and
iv. applying process discovery techniques on the summarized information.

Note that the summarized information is computed for multiples business perspectives. Each perspective is characterized by a distinct set of dimensions, and consists of a data partition on which reporting, data mining, and process mining techniques may be applied. In this chapter, we discuss the application of the Multidimensional Heuristics Miner (i.e., a process discovery technique introduced in Subsection 3.3.2) on these process perspectives. The exploitation and analysis of the multidimensional process models that can be discovered from the different process perspectives are the focus of this chapter.

Figure 5.1 positions this chapter with respect to the multidimensional process discovery approach. In the next sections, we firstly characterize the multidimensional data model as well as some OLAP concepts. Then, we introduce the Event Cube, a multidimensional data structure where the information about the different process perspectives is summarized and prepared for process discovery and analysis. Next, we describe how the information of an Event Cube can be exploited and analyzed. Finally, we discuss the challenges and issues related to the generation and exploitation of Event Cubes.

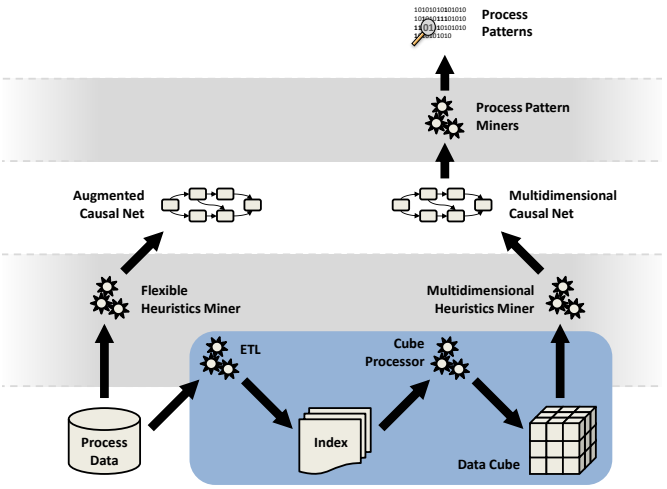


Figure 5.1: Positioning of Chapter 5 with respect to the multidimensional process discovery approach.

5.1 Multidimensional Data Model

Traditionally in OLAP, the multidimensional model represents data by means of a multidimensional *fact*-based structure that supports complex queries in real time. A fact describes an occurrence of a business operation (e.g., sale or purchase), which can be quantified by one or more *measures* of interest (e.g., the total amount of the sale or purchase) and characterized by multiple *dimensions* of analysis (e.g., time, location, product and customer). Typically numerical, measures can be aggregated for different levels of abstraction. Each dimension consists of a set of discrete values called *dimension values* or members. Eventually, there may be dimension values that represent different concept levels. For these cases, a *hierarchy* defines the order the different dimension values should be exploited, from a lower to a higher concept level. The typical example of a hierarchy using the time-based values *day*, *month*, and *year* is "$day < month < year$".

The multidimensional data model is composed by two main components: *fact tables* and *dimension tables*. Fact tables are where the facts are recorded, i.e., the measures of interest and the corresponding dimension references. The conjunction of the dimension references forms the fact ID (primary key), which identifies univocally the fact. Dimension tables are where the information about the dimensions is kept. The pair consisting of a fact table and a non-empty set of dimension tables forms the *star* schema. Eventually, in order to reduce the data storage (to the expense of the system's performance degradation), a normalization operation may be applied on a dimensional table. This means that one or more linked tables are created to hold the dimension's information. A schema with one or more normalized dimensions is called *snowflake*. Additionally, it is called *constellation* schema to the set of two or more schemas (star or snowflake) that share at least one dimension table. For further details about schemas for multidimensional databases see [50, 65].

The multidimensional data model can be implemented following three different approaches [50]. Known as multidimensional OLAP (MOLAP), the first approach relies on multidimensional data structures called *data cubes*, which maximize the system's

performance and facilitate the calculation of complex measures. The second approach, the relational OLAP (ROLAP), relies on a relational database, which theoretically solve the data storage issues. Basically, the advantages of the first approach are the disadvantages of the second. A third approach, the hybrid OLAP (HOLAP), can be eventually considered if both MOLAP and ROLAP can be combined. In this case, data that are more likely to be retrieved are maintained in a data cube, while the others are stored in a relational database.

Data Cube

Also designated as a hypercube, a data cube provides a multidimensional view of data through the materialization of specific measures for every combination of the cube's dimension values. Each combination defines a different fact and is represented by a *multidimensional cell*. A cell consists of a pair with a set of dimension values that identifies univocally the cell, and a set of aggregated values representing the measures. The set of cells that share the same dimensions forms a *cuboid*, which represents a perspective. The complete set of cuboids forms the data cube through a *lattice of cuboids*. Figure 5.2 depicts an example of a lattice of cuboids with three dimensions, which results from the combination of dimensions of a specific base table (e.g., Table 5.1).

Dimensions			
Activity	*Resource*	*Product*	*Cost*
a_1	r_2	p_3	10
a_3	r_2	p_1	15
a_3	r_1	p_1	12
a_2	r_3	p_2	15
a_1	r_3	p_2	11

Table 5.1: Example of a base table with three dimensions and one measure.

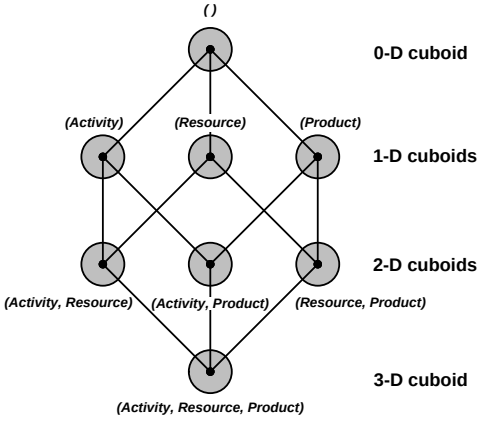


Figure 5.2: Lattice of cuboids based on the base table.

As depicted in Figure 5.2, every cuboid has a different set of dimensions. The size of this set defines the cuboid dimensionality (e.g., a 2-D or two-dimensional cuboid

is defined by two dimensions). The cuboid's cells are defined by dimension values according to the cuboid's dimensions. For instance, the top cell () belongs to the 0-D cuboid and represents all the facts in the base table (i.e., no dimension value is used as constraint). On the other hand, cell (r_3, p_2) belongs to the 2-D cuboid *(Resource,Product)*. This cell refers to all the facts in which $Resource = r_3$ and $Product = p_2$ (i.e., the last two rows of the base table). Remark that the 5 entries of the base table (Table 5.1) result in 28 cells distributed over 8 cuboids (Figure 5.3).

A cell is materialized with respect to one of its measures when all the values of the cell's representative facts – for that measure – are aggregated according a given aggregation function. For instance, using the average as an aggregation function on the measure *Cost*, the aggregation value of the top cell () is $avg(10, 15, 12, 15, 11) = 12.6$. Additionally, it is considered that a cuboid is materialized when all of its cells are materialized. The same principle applies to the data cube and its cuboids. Figure 5.3 shows the data cube after materialization of the measure *Cost*, using the average as the aggregation function.

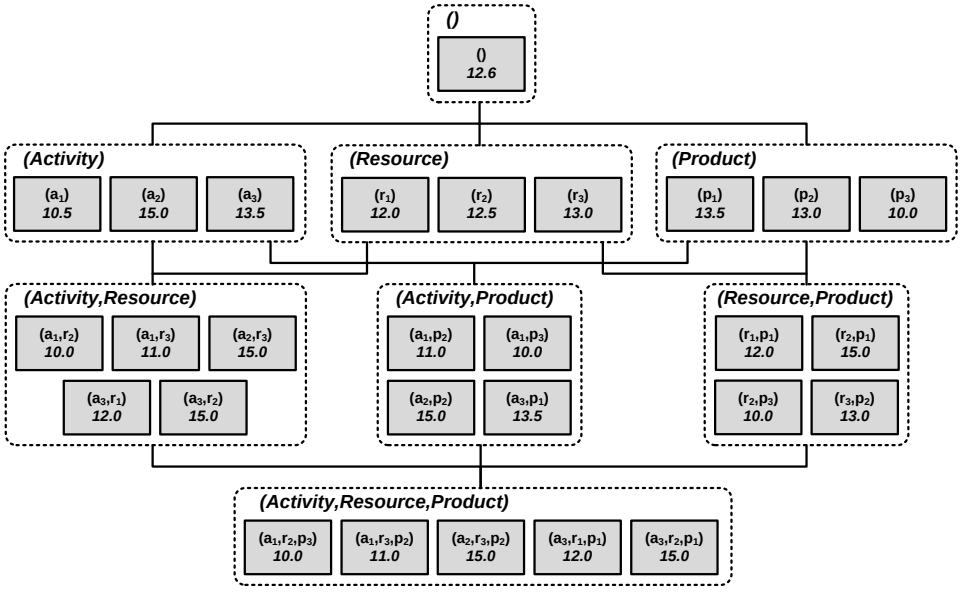


Figure 5.3: The materialized data cube.

The complexity of the data cube materialization depends on the characteristics of the aggregation functions. Theoretically, all kinds of operations can be used in the materialization process. However, the number of cells in the data cubes may constrain the usage of complex functions. Even using simple aggregation functions the materialization process can be computationally demanding. Hence, the usage of complex functions may lead to impractical materialization times. One of the solutions for this issue is the application of efficient materialization strategies. By selecting the cells to be materialized, it is possible to reduce the computational effort of the materialization process. This means that the non-selected cells need eventually to be materialized on-the-fly at the time the cell is accessed, while the others are ready to be retrieved. For an overview of materialization strategies see [50].

The analysis of a data cube consists of the exploitation of its cuboids. Moving through the lattice of cuboids, the analyst is able to adjust the analysis perspective by selecting the cube dimensions. There are five typical OLAP operators that can be used for querying multidimensional data.

Drill-Down descends one level in the lattice of cuboids by adding one dimension (or hierarchy level) to the current perspective (i.e., the level of abstraction decreases). For example, in the data cube of Figure 5.3, changing the perspective by moving from cuboid *(Activity)* to cuboid *(Activity,Resource)*.

Roll-Up ascends one level in the lattice of cuboids by removing one dimension (or hierarchy level) from the current perspective (i.e., the level of abstraction increases). For example, in the data cube of Figure 5.3, changing the perspective by moving from cuboid *(Activity,Resource)* to cuboid *(Resource)*.

Slice and Dice restricts the perspective by using filtering conditions. For example, in the cuboid *(Activity,Product)* of the data cube of Figure 5.3, considering exclusively the cells characterized by $Product = p_2$.

Pivot permutes the analysis' axes. The perspective remains the same but the information is given in a different layout.

Top-k Selection restricts the perspective to the top-k cells of a given measure. For example, in Figure 5.3, (r_2, p_1), (r_3, p_2), and (r_1, p_1) are the top-3 cells of cuboid *(Resource,Product)*.

5.2 Event Cube

An *Event Cube* is a multidimensional data structure that can hold information about different aspects (dimensions) of a business process. Like the data cubes of OLAP systems, an Event Cube can be used to improve the quality of the business analysis by providing immediate results under different levels of abstraction. Materializing business measures for all combinations of dimensions, it is possible to exploit on-the-fly the different aspects of business processes, by either analyzing directly the materialized information or applying process mining techniques on the Event Cube. Defining the process mining measurements as Event Cube measures (e.g., the necessary measurements to derive a process model), the traditional process mining applications such as process discovery can be performed according to multiple business perspectives.

5.2.1 Information Retrieval on Process Data

One of the challenges of building an Event Cube is the information retrieval on process data (i.e., searching data efficiently for computing the Event Cube information). Traditional process mining techniques search the event logs sequentially, maintaining the information about the business process in memory. So far, this strategy has proven to be effective once that the current information needs are almost static. However, multidimensional approaches have dynamic information needs. Searching the process data for specific dimension values cannot be done sequentially anymore due to efficiency issues. If filtering conditions are added to the search querying this observation becomes even more evident. The solution for this issue can be found in the information retrieval (IR) domain. Commonly used in document retrieval systems, *inverted indices*

are able to index multidimensional datasets in such a way that any dimension value can be accessed directly [145]. Basically, these indices rely on the locations where the dimension values appear in the dataset.

By applying the inverted indexing concept to process data, it is possible to retrieve all the different aspects of business processes. As stated in Section 3.1, a process instance is uniquely identified by the value PID, which is the unique key of the instance. Similarly, a process event is uniquely identified by the pair of values (PID,EID), where PID is the unique key of the process instance in which the process event was executed, and EID is the position of the process event in that process instance. Therefore, considering these identifiers, it is possible to index the characterizing attributes of process instances and events. The retrieval of these attributes depends on simple intersection operations. The intersection of sets of pair of identifiers (PID, EID) determines the locations of process events that are characterized by specific dimension values. The intersection of sets of PIDs (first element of the pair of identifiers) can be used as well to directly identify the process instances in which some events appear.

The retrieval of process events and process instances can be described as follows.

Retrieval of process events and process instances: Let W be an event log, I an inverted index on W, and C the constraint that characterizes the process events to be retrieved. For each constraint value $D{:}d \in C$, $tid^E(D{:}d)$ denotes the set of process event identifiers (i.e., pairs of values (PID,EID)) of the process events that satisfy the dimension value d of the dimension D. The corresponding set of process instance identifiers ($tid^I(D{:}d)$) can be derived from $tid^E(D{:}d)$ by simply considering the first value (PID) of the process event identifier. The intersection of all $tid^E(D{:}d)$ sets of C (i.e., $tid^E(C) = \bigcap_{(D:d)\in C}[tid^E(D{:}d)]$) determines the identifiers of the process events characterized by C. Based on the identifiers, the process event retrieval can be performed by directly accessing the process events in the event log. The intersection of all $tid^I(D{:}d)$ sets of C (i.e., $tid^I(C) = \bigcap_{(D:d)\in C}[tid^I(D{:}d)]$) determines the identifiers of the process instances characterized by C. Based on the identifiers, the process instance retrieval can be performed by directly accessing the process instances in the event log.

The repair process of Figure 5.4 is introduced to illustrate the retrieval of process events. The goal of this process is to fix malfunctioning products. There may be more than one repair attempt but not unsuccessful process cases (i.e., the end of a process instance implies that the product is fixed). Two aspects of this process are registered in an event log: the performed activities (dimension *Activity*) and information produced in one of the activities about the problem type (dimension *Problem Type*).

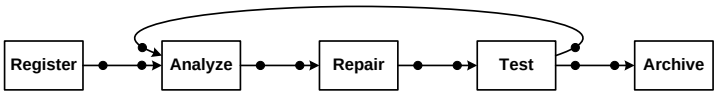


Figure 5.4: An example of a repair process.

Figure 5.5 describes four process instances of the repair process of Figure 5.4. Let us assume that these are the first four process instances in the event log. Remark that the problem type (information produced in activity *Analyze*) is identified in brackets. Although this information is given as an event-based attribute of activity *Analyze*), the problem type could also be used to characterize other activities in the process

instance. Event-based attributes that characterize not only the process event where the attribute is described but also other process events in the same process instance (e.g., dimension *Problem Type*) are designated as *false event-based attributes*.

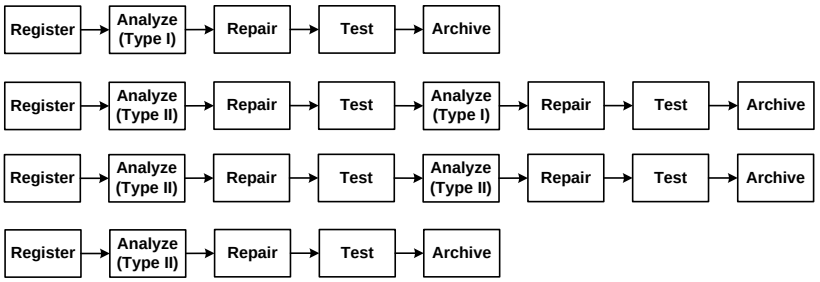


Figure 5.5: Four process instances of the repair process of Figure 5.4.

Table 5.2 presents the inverted index of the process instances in Figure 5.5. Remark that the PID is defined by the process instance's position in the figure (from top to bottom). Table 5.2a describes the event locations of the different activities in the repair process. Similarly, Table 5.2b describes in which process events the different problem types are known.

Activity	*Event Locations*
Register	$\{(1,1),(2,1),(3,1),(4,1)\}$
Analyze	$\{(1,2),(2,2),(2,5),(3,2),(3,5),(4,2)\}$
Repair	$\{(1,3),(2,3),(2,6),(3,3),(3,6),(4,3)\}$
Test	$\{(1,4),(2,4),(2,7),(3,4),(3,7),(4,4)\}$
Archive	$\{(1,5),(2,8),(3,8),(4,5)\}$

(a) Index of the dimension *Activity*.

Problem Type	*Event Locations*
Type I	$\{(1,2),(2,5)\}$
Type II	$\{(2,2),(3,2),(3,5),(4,2)\}$

(b) Index of the dimension *Problem Type*.

Table 5.2: Inverted index of the process instances in Figure 5.5.

The retrieval of all process events (and process instances) in which an analysis is performed and the problem type is identified as *Type II* can be described as follows. $C = \{Activity{:}Analyze, Problem\ Type{:}Type\ II\}$ is the constraint that characterizes the process events to be retrieved. According to Table 5.2a, the set of identifiers of the process events that satisfy the dimension value *Analyze* of the dimension *Activity* is $tid^E(Activity{:}Analyze) = \{(1,2),(2,2),(2,5),(3,2),(3,5),(4,2)\}$. Analogously, according to Table 5.2b, $tid^E(Problem\ Type{:}Type\ II) = \{(2,2),(3,2),(3,5),(4,2)\}$ is the set of process event identifiers for the dimension value *Type II* of the dimension *Problem Type*. The corresponding sets of process instance identifiers can be derived from the sets of process instance identifiers by simply considering the first value (PID) of the process event identifier. Therefore, $tid^I(Activity{:}Analyze) = \{1,2,3,4\}$ and

$tid^I(Problem\ Type{:}Type\ II) = \{2, 3, 4\}$. The exact locations where process events characterized by both activity *Analyze* and problem type $Type\ II$ appear in the event log are given by:

$$\begin{aligned} tid^E(C) &= \{(1,2),(2,2),(2,5),(3,2),(3,5),(4,2)\} \cap \{(2,2),(3,2),(3,5),(4,2)\} \\ &= \{(2,2),(3,2),(3,5),(4,2)\} \\ tid^I(C) &= \{1,2,3,4\} \cap \{2,3,4\} = \{2,3,4\} \end{aligned}$$

The value $(2, 2)$ of $tid^E(C)$ denotes that the second process event of the second process instance is both activity *Analyze* and problem type $Type\ II$. Analogously, $tid^I(C)$ indicates that process events characterized by both activity *Analyze* and problem type $Type\ II$ appear in three process instances: the second, the third, and the fourth.

Event-Based Attributes

One of the issues of retrieving information on process data is that the event-based attributes used to identify instance information, the so-called false event-based attributes, are not handled properly. This often happens because the information is not known until the execution of a specific activity in the process. Hence, instead of being associated to process instances, the information is given as a specific aspect of a process event. A good example is the repair process of Figure 5.4 in which the problem type is only known after some analysis (i.e., in the activity *Analyze*). Although the information is given as a characteristic of the activity *Analyze*, the problem type can be extended to other activities of the process instance.

Table 5.2b describes in which process events the different problem types are known. This indexing would be correct if the problem type information was a true event-based attribute. However, since the problem type is a false event-based attribute, the inverted index on this kind of attribute needs to be built in a different way. Instead of describing in which process events a specific attribute appears, the index should identify which process instances the attribute is associated to. In other words, all process events of the process instance where the attribute appears must be characterized by that information. This can be achieved by normalizing the index of the false event-based attribute. This normalization consists of changing, in the index, the EID values to the value '$*$'. This wildcard value represents all the positions of a process instance. For cases where a false event-based attribute is defined multiple times in the same process instance, a new attribute value is created by aggregating all the available information in such a way that no information is lost and the results consistency is preserved. For example, both third and fourth process instances of Figure 5.5 are characterized by problems of type II. However, these process instances are distinct once that in one of the instances only one repair is performed, while in the other there are two repair activities. Applying a summarization function (e.g., the average throughput time) on both process instances (and also on the second process instance of Figure 5.5 once that it is also partially characterized by problems of type II) would lead to results inconsistency. This issue is even more evident if the first and second process instances of Figure 5.5 are taken into account as being characterized by problems of type I. Table 5.3 shows the result of this normalization on the index of Table 5.2b.

Let us use the following example to illustrate the false event-based attribute issue. The retrieval of all process events in which a repair is performed on a problem of type

Problem Type	*Event Locations*
$Type\ I$	$\{(1,*)\}$
$Type\ II\ >\ Type\ I$	$\{(2,*)\}$
$Type\ II\ >\ Type\ II$	$\{(3,*)\}$
$Type\ II$	$\{(4,*)\}$

Table 5.3: Inverted index of the process instances in Figure 5.5.

I (i.e., event occurrence $\{Activity{:}Repair, Problem\ Type{:}Type\ I\}$) can be described as follows. Considering the problem type information as a true event-based attribute issue, the location of the process events are defined by the intersection of identifiers from tables 5.2a and 5.2b. The result is that no process event is characterized by these two attributes (*Activity* and *Problem Type*), i.e., $\{(1,3),(2,3),(2,6),(3,3),(3,6),(4,3)\} \cap \{(1,2),(2,5)\} = \emptyset$. Considering the problem type information as a false event-based attribute issue, the location of the process events are defined by the intersection of identifiers from tables 5.2a and 5.3. This time the third process event of the first process instance is given as a result once that $\{(1,3),(2,3),(2,6),(3,3),(3,6),(4,3)\} \cap \{(1,*)\} = \{(1,3)\}$. Remark that, although the second process instance contains a repair activity on a problem of type I (i.e., the process event with the identifier $(2,6)$), this process event is not included in the results because there is more than one problem type in the same instance.

5.2.2 Deriving Process Information

Two types of process information can be represented in an Event Cube:

- **Characterizing information** describes a specific characteristic of a process instance or a process event. For example, the activity label describes which activity was executed in a specific process event.
- **Measuring information** quantifies a specific measurement of a process instance or a process event. For example, the frequency of a process event quantifies how many times the event occurred in the process execution.

For both types, the process information needs to be retrieved from the data sources and, eventually, transformed to meet specific requirements. The retrieval operation is described in the previous subsection. The transformation operation consists of any sort of function that converts process information with low analytic potential into information that can enrich the process analysis.

Numerical Information

Numerical information is a natural candidate for being analyzed as measuring information. Nonetheless, there are situations where the numerical information needs to be prepared for analysis. For example, sometimes it is convenient to remove outliers from the dataset, which can be achieved by applying some outlier detection method [17]. Other common numerical transformations are interpolation (and extrapolation) and normalization.

Interpolation aims at the estimation of missing values in the dataset. For example, let us consider the following sequence of numerical values: $[0.8, 2.1, 3.4, ?, 4.8, 5.1]$.

By applying the linear interpolation method, which in this case basically consists of estimating the average value of the preceding and following values, it is possible to estimate that ? = (3.4 + 4.8)/2 = 4.1. Several other interpolation methods can be found in the literature [82]. Similar to interpolation, extrapolation methods estimate (forecast) the next values of a sequence. For an overview of extrapolation methods see [102].

Also known as standardization, normalization aims at the transformation of multiple attributes by converting their different domains into a common domain. This transformation facilitates the comparison of numerical information with different domains. For example, let us consider the sequences A_1, A_2, and A_3 described in Table 5.4. These sequences represent time series of data from three sensors of a specific machine.

Attribute	*Domain*	*Values*
A_1	$[0, 1]$	0.59, 0.61, 0.58, 0.57, 0.29, 0.47, ...
A_2	$[5, 95]$	22.1, 18.9, 26.7, 31.9, 85.1, 63.4, ...
A_3	$[-5, 5]$	3.1, 2.5, 1.2, 2.4, −1.0, −0.1, ...

Table 5.4: Example of three numerical attributes with different domains.

Since the domains of A_1, A_2, and A_3 are not the same, the comparison of the different time series is not straightforward. Three simple normalization methods can be considered to overcome this issue:

- **Standard Score** (Z-Score) indicates how many standard deviations a value is above or below the mean (i.e., $v' = (v - \mu)/\sigma$, where μ and σ are the attribute's arithmetic mean and standard deviation). For example, by z-score normalization and assuming the values $\mu = 0.52$ and $\sigma = 0.21$, the normalized values of attribute A_1 of Table 5.4 are: 0.33, 0.43, 0.29, 0.24, −1.10, −0.24, etc.
- **Min-Max** is based on the minimum and maximum values. By replacing each value v by $v' = (v - min)/(max - min)$, it is possible to transform the attribute's domain $[min, max]$ into $[0, 1]$. For example, by min-max normalization, the normalized values of attribute A_2 of Table 5.4 are: 0.19, 0.15, 0.24, 0.30, 0.89, 0.65, etc.
- **Decimal Scaling** is based on the division of the values by 10^x (i.e., $v' = v/10^x$), where x is the smallest integer for which the absolute maximum value is less than 1 (i.e., $|v'| < 1$). For example, for attribute A_3 of Table 5.4, $x = 1$ is the smallest integer for which the absolute maximum value (i.e., $|-5/10^1| = |5/10^1| = 0.5$) is less than 1. Hence, by decimal scaling normalization, the normalized values of A_3 are: 0.31, 0.25, 0.12, 0.24, −0.10, −0.01, etc.

There are situations where numerical information is rather characterizing information. A good example for this observation is the attribute *age*, which characterizes the age of a specific individual or object. Like in many mining techniques, most numerical attributes cannot be used as characterizing information because of their continuous nature. The role of the characterizing information is to group the data by similar characteristics. For example, in control-flow mining, the attribute *Activity* is used as characterizing information for grouping the process events by activity. The number of groups of process events is defined by the number of distinct activity values (i.e., the

cardinality of the attribute *Activity*). Depending on the attribute's distinct values, using a numerical attribute as characterizing information can lead to the generation of an excessive number of groupings. This means that the representation and analysis of a numerical attribute can be challenging.

The discretization aims at the transformation of a numerical (continuous) attribute into a categorical (discrete) one. This is achieved by dividing the range of the attribute into categories (e.g., intervals). By replacing numerical values by categories, the number of distinct values can be reduced considerably, facilitating thus the attribute analysis. There are two categories of discretization methods: *supervised* and *unsupervised*. On the one hand, supervised discretization methods do not consider only the numerical values to be discretized but also reference values. Also known as classes, the reference values can be used to supervise (assist) the distribution of numerical values across the different categories. For example, the *Chi2* method uses statistical significance tests – on the relationship between a group of numerical values and a specific class – to optimize the grouping of the numerical values. Other common supervised methods such as *MDLP* and *ID3* are based on entropy measures (further details about entropy are presented in Subsection 6.2.4). On the other hand, unsupervised discretization methods simply consider the set of numerical values to be discretized. Basically, this category of discretization method distribute equally the numerical values across a given number of bins (categories) k. For example, the *equal-width* method divides the attribute's domain into k equal-width intervals, representing each interval as a bin. Another common binning method is the *equal-frequency* in which k bins with equal number of values are created. For an overview of discretization methods see [75].

Time Information

Time information (a timestamp) is typically considered as characterizing information. Nonetheless, like numerical information, time information is characterized by a continuous nature, which means that some kind of *time discretization* may be necessary to reduce the number of time references. By time discretization we mean the division of timestamps attributes by the existing time categories (e.g., day of week). Therefore, by applying simple binning-based methods on timestamps, it is possible to transform the time information into categorical information. An example of time discretization is described as follows. Let us assume that the original timestamp is 19-06-2012:15.45. Some possible time categories into which the given timestamp can be transformed are the following:

- **Hour** is the hour part of the timestamp. For example, 15 is the hour of 19-06-2012:15.45.
- **Date** is the date part of the timestamp. For example, 19-06-2012 is the date of 19-06-2012:15.45.
- **Day of Month** is the day part of the date. For example, 19 is the day of month of 19-06-2012.
- **Day of Week** is the date's day of week according to the Gregorian calendar. For example, *Tuesday* is the day of week of 19-06-2012.
- **Month** is the month part of the date. For example, 06 or *June* is the month of 19-06-2012.

- **Quarter** is the quarter (three month period) of the date. For example, $Q2$ is the quarter of 19-06-2012.
- **Year** is the year part of the date. For example, 2012 is the year of 19-06-2012.

Categorical Information

Naturally, categorical information can be considered as characterizing information. Nevertheless, there are situations where the categorical information can be used to derive extra information that can enrich the process analysis. For instance, the attribute *Originator* may be used to derive information about the team and department the resource is part of. Thus, not only the performance of the resource can be assessed but also the performance of its team and department. Obviously, this kind of transformation only can be achieved if

i. there is a specific transformation function capable of converting some given information into another (according to some requirements), and

ii. there is enough information in the data sources on which the transformation function can be applied.

5.2.3 Measures of Interest

A measure of interest consists of measuring information about a specific aspect of the business process. This information can be used to summarize the process behavior represented in a multidimensional process model. Hence, a measure can be categorized as follows.

Instance-Based Measures quantify a specific aspect of the business process by considering process instance-related information. For example, the average throughput time of all process instances.

Event-Based Measures quantify a specific aspect of the business process by considering process event-related information. For example, the average execution time of process events.

Flow-Based Measures quantify a specific aspect of the business process by considering information about causal dependencies. For example, the average waiting time between process events.

Binding-Based Measures quantify a specific aspect of the business process by considering information about splits and joins. For example, the number of times a split (output binding) was activated.

The aggregation of the information from a given measure can provide useful insight into the process behavior. Aggregation functions can be applied on measures in order to summarize the measuring information, facilitating thus the process analysis.

In an Event Cube, measures of interest can have two distinct scopes: (i) process discovery, and (ii) performance analysis. Measures that facilitate the discovery of process models are designated **control-flow measures**. Measures that support the performance analysis of processes are designated **performance measures**. Remark that a measure of interest can be both a control-flow and a performance measure (e.g., the event occurrence frequency).

Control-Flow Measures

A *control-flow measure* consists of measuring information about a specific aspect of the business process, which can used by a control-flow algorithm in the discovery of process models. For example, the basic multidimensional relations (cf. Definition 3.15) can be considered as control-flow measures. The materialization of control-flow measures in the Event Cube enables the execution of process discovery techniques over any process perspective. Moreover, since the measuring information is computed beforehand, the process discovery – and analysis – on the Event Cube can be performed efficiently and on-the-fly.

The Multidimensional Heuristics Miner relies on the following control-flow measures.

- **Event-Based Measures**
  - **Event Entry**: the set of process event identifiers (PID, EID) of an event occurrence. The number of elements of this set represents the event occurrence frequency.
  - **Start Event**: the set of process event identifiers of a given event occurrence in which EID = 1 (i.e., the first position of the process instance). The number of elements of this set represents the frequency of the event occurrence as a start event.
  - **End Event**: the set of process event identifiers of a given event occurrence in which EID = n, with n the last position of the process instance. The number of elements of this set represents the frequency of the event occurrence as an end event.
- **Flow-Based Measures**
  - **Direct Successor**: for each other cuboid's event occurrence b, the set of process event identifiers of a given event occurrence a in which its EID is one position before the EID of the other. The number of elements of each set represents the number of times the given event occurrence is a predecessor of the other (i.e., the number of times the $a >_W b$ measurement of Definition 3.15 occurs in the process data).
  - **Length-Two Loop**: for each other cuboid's event occurrence b, the set of process event identifiers of a given event occurrence a in which its EID is one position before the EID of the other (i.e., direct successor), and two positions before another EID of itself. The number of elements of each set represents the number of times the given event occurrence has a length-two loop via other occurrence (i.e., the number of times the $a \gg_W b$ measurement of Definition 3.15 occurs in the process data).
  - **Direct Successor Dependency**: for each other cuboid's event occurrence, the dependency measure $a \Rightarrow_W b$ between the given event occurrence a and another b (Definition 3.13 applied on event occurrences instead of activities).
  - **Indirect Successor Dependency**: for each other cuboid's event occurrence, the dependency measure $a \Rightarrow^2_W b$ between the given event occurrence a and another b (Definition 3.13 applied on event occurrences instead of activities).

Remark that the control-flow measures presented in this subsection are the necessary measurements to discover a process model using the Multidimensional Heuristics Miner. Other control-flow measures may be defined and used in the framework in order to facilitate the execution of other control-flow algorithms on the Event Cube.

Table 5.5 presents an overview about the necessary measures to build a multidimensional dependency graph. The dependencies between measures define the order by which the measures should be materialized in order to optimize the materialization process. For instance, measure *Start Event* is a subset of *Event Entry*. So, making use of the materialized information of *Event Entry*, it is not necessary to compute – again – the set of process event identifiers of a given cell. Using the *Event Entry*, it is only necessary to identify the process event identifiers that fulfill the *Start Event* definition (i.e., all identifiers with EID = 1).

Type	*Measure*	*Dependencies*
Event-Based	Event Entry	
Event-Based	Start Event	Event Entry
Event-Based	End Event	Event Entry
Flow-Based	Direct Successor	Event Entry
Flow-Based	Length-Two Loop	Event Entry
Flow-Based	Direct Successor Dependency	Direct Successor
Flow-Based	Indirect Successor Dependency	Length-Two Loop

Table 5.5: Overview of the control-flow measures used in the Multidimensional Heuristics Miner.

Performance Measures

A *performance measure* consists of measuring information about a specific aspect of the business process that can be used to get insight into the performance of the business process. For example, the average throughput time of the process can be considered as a performance measure. The materialization of performance measures in the Event Cube enables the analysis of the business process.

Some common measurements used in process discovery techniques can be described as performance measures.[1]

- **Instance-Based Measures**
  - **Instance Entry**: the set of process instance identifiers (PID) of an event occurrence. The number of elements of this set represents the number of times the event occurrence appears in distinct instances.
  - **Throughput Time**: the set of time periods of an event occurrence that represent the throughput times of the process instances where the given occurrence appears. Computing this measure requires temporal information (timestamps) of process events, and consists of the timestamp difference between the instance's first and last events.
  - **Instance Value**: the set of values of an event occurrence provided by a specific instance-based attribute.

[1] These process discovery techniques can be found in the ProM framework [123, 125].

  - **Event Value Aggregation**: the set of values of an event occurrence provided by the aggregation of all information about a specific event-based attribute described in a process instance.

- **Event-Based Measures**

  - **Sojourn Time**: the set of time periods that represent the sojourn times of an event occurrence. Temporal information (timestamps) of process events regarding the event execution (i.e., the information provided by the attribute *Event Type*) is required for computing this measure. The sojourn time of an event occurrence is defined as the timestamp difference between the occurrence's completion and start.
  - **Inactive Time**: the set of time periods that represent the inactive times of an event occurrence. Like for the Sojourn Time, this measure requires temporal information (timestamps) of process events regarding the event execution. The inactive time of an event occurrence is defined as the sum of the time periods the occurrence was in a state of suspension (i.e., all the pauses in the occurrence execution).
  - **Execution Time**: the set of time periods that represent the execution times of an event occurrence. The execution time can be defined as the sojourn time minus the inactive time.
  - **Event Value**: the set of values of an event occurrence provided by a specific event-based attribute.

- **Flow-Based Measures**

  - **Waiting Time**: for each other cuboid's event occurrence b, the set of time periods that represent the waiting times between a given event occurrence a and b. As any other time-related measure, this measure depends on temporal information (timestamps) of process events regarding the event execution. The waiting time is defined as the timestamp difference between the completion time of a and the start time of b.
  - **Delta Value**: for each other cuboid's event occurrence b and according to a specific event- or instance-based measure, the set of delta values between a given event occurrence a and b.

- **Binding-Based Measures**

  - **Binding Entry**: the set of process event identifiers (PID, EID) of an event occurrence binding. The number of elements of this set represents the binding frequency.

Remark that, besides the measures described above, other performance measures may also be considered.

Aggregation Functions

An *aggregation function* can be defined as a function that transforms a collection of objects into a single object that summarizes the collection according to a specific criteria. By a collection of objects we mean a group of objects that can be represented

either in a list or in a set. The outcome value of an aggregation function is designated as an *aggregated value*.

An aggregation function can be categorized as follows [50].

Distributive functions are those which can be computed by partitioning the collection of objects. Assuming that the collection is divided in n partitions, the result of the distributed application of the function on the different partitions consists of n aggregated values. The function is distributive if the aggregation of the collection of objects can be computed from these n aggregated values For example, the functions *sum* and *count* can be computed under these circumstances.

Algebraic functions are those which can be computed by combining a finite number (bounded positive integer) of distributive functions. For example, the function *average* can be computed by combining the distributive functions *sum* and *count* (i.e., $average = sum/count$).

Holistic functions are those for which it is not possible to determine a constant number of distributive functions needed to characterize the function computation. The functions *median* and *mode* are common examples of holistic functions.

Different types of aggregation functions are introduced next. Remark that it is not intended to present a complete list of the aggregation functions that can be used on the Event Cube but rather a list of potentially interesting and useful functions to be used in process analysis. Therefore, any other aggregation function not mentioned in this list can also be used in the Event Cube.

Generic Aggregation Functions: An aggregation function that can work over any collection of objects is designated as a *generic aggregation function*. Common examples of this type of aggregation function are the following.

- **Count**: determines the number of elements of a collection of objects.
- **Contain**: checks whether a given object is in a collection of objects.
- **Mode**: identifies the most frequent object of a collection of objects.
- **Median**: identifies the object that separates the lower and higher halves of a collection of objects.[2]

Numerical Aggregation Functions: An aggregation function that can work over a collection of numerical values is designated as a *numerical aggregation function*. Common examples of this type of aggregation function are the following.

- **Sum**: adds all elements of a collection of numerical values.
- **Min**: identifies the minimum value of a collection of numerical values.
- **Max**: identifies the maximum value of a collection of numerical values.
- **Average**: calculates the arithmetic mean of a collection of numerical values.
- **Standard Deviation**: estimates the standard deviation of a collection of numerical values.
- **Variance**: estimates the variance of a collection of numerical values.
- **Product**: multiplies all elements of a collection of numerical values.

[2] Computing the median is only possible if the collection of objects can be sorted.

Time Aggregation Functions: An aggregation function that can work over a collection of timestamps is designated as a *time aggregation function*. Common examples of this type of aggregation function are the following.

- **Begin**: identifies the earliest temporal reference of a collection of timestamps.
- **End**: identifies the latest temporal reference of a collection of timestamps.
- **Period**: calculates the time period of a collection of timestamps (i.e., the difference between the latest and earliest timestamps).
- **Rate**: determines the occurrence rate of the elements of a collection of timestamps (i.e., the number of elements in the collection divided by the difference between the latest and earliest timestamps).

Map-Based Aggregation Functions: Typically, an aggregation function works over a collection of objects. For example, the frequency of an event occurrence can be determined by aggregating its process event identifiers according to the function *Count*. However, there are cases in which more complex data structures are needed to describe a specific aspect of the case. For example, an event occurrence is caused (and followed) by zero or more event occurrences. Hence, the process event identifiers of the causes of an event occurrence cannot be determined by a single set of identifiers. Instead, the different sets of identifiers should be grouped by cause. Therefore, a new data structure should be considered.

A map is a data structure that maps unique keys to values. Considering a value as a collection of objects, a map can be used to group collection of objects by specific relationships (e.g., the different causes of an event occurrence). The application of an aggregation function on a map consists of applying that function on every collection of objects in the map. As the result of the aggregation, it is produced a map of aggregated values keeping the key mappings. In this way, it is possible to use any set- or list-based aggregation function.

Table 5.6 presents an overview of the aggregation functions introduced in this subsection, where every function is characterized in terms of category. Distributive functions should be preferred over the others in order to optimize the materialization process. These functions can be computed efficiently because their computation processes can be partitioned. On the other extreme, holistic functions should be avoided because of their complex computation processes.

5.2.4 Materializing the Event Cube

An Event Cube is defined as a data cube of events and can be used to extend the current process mining techniques with multidimensional capabilities. Based on process data, the Event Cube organizes the data in such a way that is possible to analyze the business information under different levels of abstraction. Applying the *Shell Cube* materialization strategy [73], i.e., only a selection of dimensions of interest is materialized, an Event Cube can be built to accommodate all the necessary measurements to perform process discovery and analysis. This means that, all the information regarding the different perspectives (and levels of abstraction) is computed and maintained in the cube, facilitating thus the execution of all types of – business – queries.

Type	*Aggregation Function*	*Category*
Generic	Count	Distributive
Generic	Contain	Distributive
Generic	Mode	Holistic
Generic	Median	Holistic
Numerical	Sum	Distributive
Numerical	Min	Distributive
Numerical	Max	Distributive
Numerical	Average	Algebraic
Numerical	Standard Deviation	Algebraic
Numerical	Variance	Algebraic
Numerical	Product	Distributive
Time	Begin	Distributive
Time	End	Distributive
Time	Period	Distributive
Time	Rate	Algebraic

Table 5.6: Overview of aggregation functions.

The materialization process of an Event Cube can be described as follows. Relying on an index instead of directly on the process data (e.g., event log), the lattice of cuboids according to a given set of dimensions is firstly built. Representing a different perspective, each of these cuboids holds the event occurrences that characterize the perspective. These occurrences are represented as multidimensional cells and can be characterized by multiple measures. The Event Cube materialization process is finished when all the considered measures are computed for every cell in the cube. Computing a measure for a multidimensional cell can be defined as follows.

i. Using the inverted index, compute the set of identifiers of the process events characterized by the dimension values of the cell.

ii. Compute the measure value for every process event in the set of identifiers. This can be achieved either by retrieving a specific attribute of the process event or by applying a specific transformation function over the set of identifiers.

iii. Apply an aggregation function to summarize the collection of measure values computed in the previous step.

Finally, the materialized measures can be either directly analyzed (e.g., using a pivot table) or used for process mining (e.g., deriving multidimensional process models). All of these steps are summarized in Algorithm 6.

Considering $dimensions = \{A, B, C\}$ and $measures = \{m_1, m_2\}$ as the set of dimensions and measures to be materialized, each step of the materialization process of an Event Cube (i.e., each line of Algorithm 6) is described as follows.

1. The data structure that represents the lattice of cuboids of the Event Cube is initialized.
2. Every possible combination of dimensions is described in the powerset of $dimensions$ ($\{A, B, C\}$). Hence, $X = \{ \emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\} \}$. Every element of X will be used to characterize a different cuboid in the Event Cube.

Algorithm 6: Materializing the Event Cube

Input : The inverted index and the sets of dimensions and measures to be materialized.

Output: The materialized Event Cube.

Method

```
1  init lattice;
2  X ← the powerset of dimensions;
3  foreach set of dimensions D ∈ X do
4  |  cuboid ← buildCuboid(index, D, measures );
5  |  add cuboid to lattice;
   end
6  return lattice;
```

function buildCuboid(index, D, measures)

```
a  cuboid ← new cuboid characterized by the dimensions in D;
b  Y ← the combination of dimension values of the dimensions in D such that
   every combination contains exactly one value from every dimension in D;
c  foreach set of dimension values V ∈ Y do
d  |  cell ← new cuboid cell characterized by V;
e  |  TID ← retrieve, using index, the set of identifiers (PID, EID) of the process
   |  events characterized by the dimension values of V;
f  |  foreach measure m ∈ measures do
g  |  |  value ← compute m for the process events identified in TID;
h  |  |  add value to cell as measure value of m;
   |  end
i  |  add cell to cuboid;
   end
j  return cuboid;
```

3-5. For each combination of dimensions D in X, a new cuboid – characterized by the dimensions in D – is created (and materialized) through the function *buildCuboid*(), and added to the lattice. Further details about this function are provided below.

6. The lattice of cuboids of the Event Cube is generated. Since all cuboids in the lattice are materialized, the lattice of cuboids can be returned as the materialized Event Cube.

The function *buildCuboid*() creates a cuboid characterized by a given set of dimensions (D), generates all cuboid's cells according to the values of the dimensions in D, and computes – for every cell – a given set of measures (*measures*). Let $D = \{A, C\}$ be the set of dimensions to characterize the cuboid, $A = \{a_1, a_2, a_3\}$ and $C = \{c_1, c_2\}$ the sets of dimension values of A and C, $measures = \{m_1, m_2\}$ the set of measures to be materialized, and m_1 and m_2 the average cost and the *Start Event* control-flow measure introduced in Subsection 5.2.3. Each step of function *buildCuboid*() is described as follows.

a. The data structure that represents the cuboid is initialized.

b. The combination of dimension values of the dimensions in D (i.e., $\{A, C\}$) is computed such that every combination contains exactly one value from dimensions A and C. Hence, $Y = \{ \{a_1, c_1\}, \{a_1, c_2\}, \{a_2, c_1\}, \{a_2, c_2\}, \{a_3, c_1\}, \{a_3, c_2\} \}$.

c-i. For each combination of dimension values V in Y, a new cuboid's cell – characterized by the dimension values in V, is created (line d), materialized (lines e-h), and added to the cuboid (line i). Materializing a cell consists of computing the measures through the set of identifiers of the process events characterized by the cuboid's cell. For example, let us assume that $TID = \{(1, 1), (2, 3), (3, 5)\}$ is the set of identifiers of the process events characterized by the cuboid's cell (i.e., event occurrence). For measure m_1, the average cost of the process events characterized by the cell (assuming that this information is available), each identifier in TID is used to retrieve the cost information of the corresponding process event. The average of the retrieved costs defines the final value of m_1. For measure m_2, the *Start Event* control-flow measure, the set of identifiers TID is filtered in such a way that only the process events with $EID = 1$ remain in the set. The number of remaining elements in the set defines the final value of m_2 (in this example, $m_2 = 1$). The multidimensional cell is added into the cuboid when all measures are computed.

j. The cuboid is generated and materialized. Since all cuboid's cells are materialized, the cuboid is ready to be returned as result.

5.3 Multidimensional Process Analysis

Multidimensional process analysis consists of exploiting the Event Cube in order to get insight into the business process. This is achieved by performing multidimensional process discovery based on specific business process queries.

5.3.1 Multidimensional Process Discovery

Traditional process discovery consists of extracting a – traditional – process model from some given process data (e.g., event log). These techniques describe the process behavior with respect to typical process perspectives such as the one that simply describes the performed activities. Changing the focus of traditional process discovery techniques to non-typical process perspectives is not possible in most cases, although it may be achieved indirectly by manipulating the process data.

Multidimensional process discovery consists of extracting – multidimensional – process models from an Event Cube. These techniques are capable of describing the business process according to any process perspective supported in the process data. Furthermore, the process behavior can be characterized by one or more aspects of the business process. By materializing the necessary measurements to perform process discovery, it is possible discover on-the-fly – and in an integrated environment – the different perspectives of the business process.

An Event Cube consists of a lattice of cuboids. Representing a distinct process perspective, each cuboid is formed by multidimensional cells which identify specific groups of process events described in the process data. Each cell is defined by a distinct combination of dimension values according to the dimensions of the cuboid that contains the cell. Additionally, after materialization, every cell in the Event Cube is characterized

by measuring information (e.g., the control-flow measures). Multidimensional process discovery relies on this information to describe the process behavior with respect to some process perspective. By applying some OLAP-based operations (e.g., drill-down and roll-up), the focus of the process discovery can be adjusted on-the-fly in order to fulfill specific interests or requirements.

The following example is presented to illustrate how process discovery can be performed on an Event Cube. Let us consider an Event Cube with two dimensions: *Activity* and *Resource* (with $values(Activity) = \{A, B, C, D\}$ and $values(Resource) = \{W, X, Y, Z\}$). The lattice of cuboids of this Event Cube is presented in Figure 5.6.

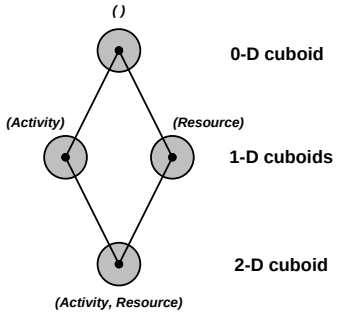


Figure 5.6: Lattice of cuboids based on the dimensions *Activity* and *Resource*.

Characterized by a distinct combination of dimension values, the multidimensional cells that form the different cuboids of Figure 5.6 are characterized as follows.

- the multidimensional cell () is the only cell of the 0-D cuboid *()*. Being characterized by no dimension, this cell represents all process events described in the process data.
- four multidimensional cells form the 1-D cuboid *(Activity)*: (A), (B), (C), and (D). Cell (A), for instance, represents all process events in which activity A was performed.
- four multidimensional cells form the 1-D cuboid *(Resource)*: (W), (X), (Y), and (Z). Cell (W), for instance, represents all process events performed by W.
- sixteen multidimensional cells form the 2-D cuboid *(Activity,Resource)*: (A,W), (A,X), ..., (D,Y), and (D,Z). Cell (A,W), for instance, represents all process events in which activity A was performed by W. Remark that some of these cells may have no support in the process data.

In order to facilitate the application of process discovery on the Event Cube, the control-flow measures must be materialized for every multidimensional cell. Thus, since the necessary measurements for performing process discovery are computed beforehand, the discovery can be performed on-the-fly over different business perspectives. Figure 5.7 shows part of the materialized Event Cube of the running example.

In this example, process discovery can be performed taking into account four different process perspectives. From the cuboids that represent these perspectives, nine distinct multidimensional process models can be generated through the materialized information and the Multidimensional Heuristics Miner. These multidimensional process models can be described as follows.

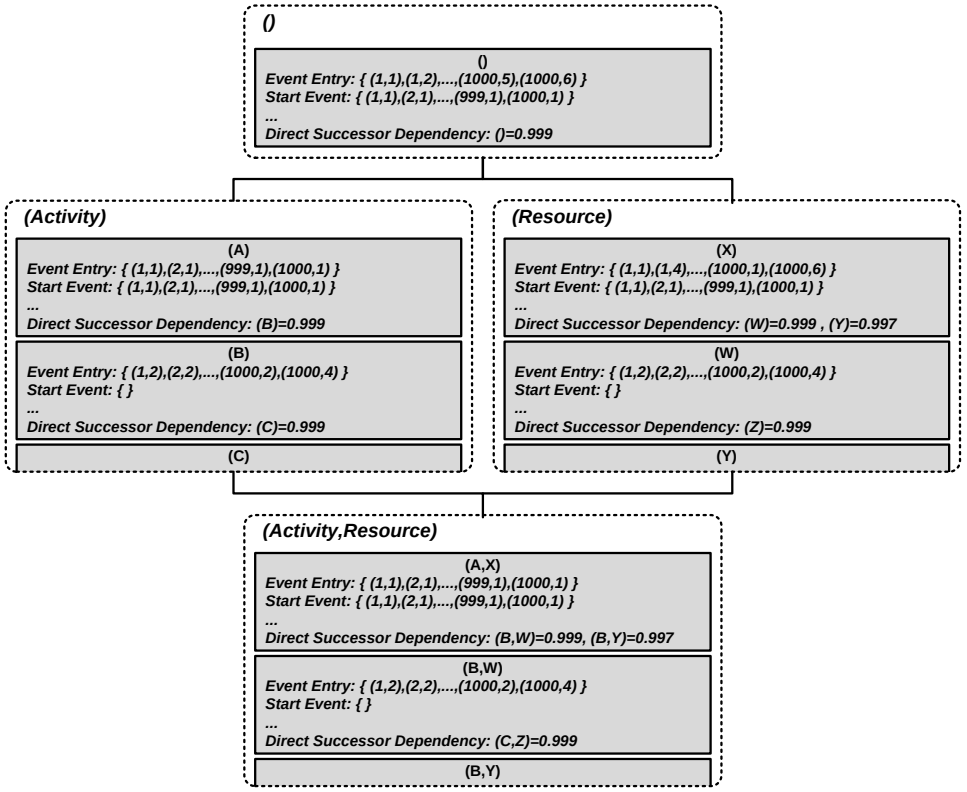


Figure 5.7: The materialized Event Cube.

- **0-D Cuboid** *()* represents the most abstract process perspective. Since this perspective is characterized by no dimension, the only possible multidimensional process model that can be generated from this cuboid is the one with 0-D nodes and 0-edges (Figure 5.8).

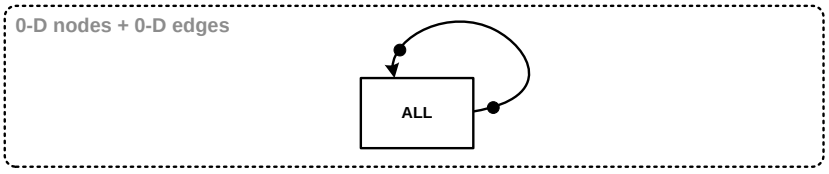
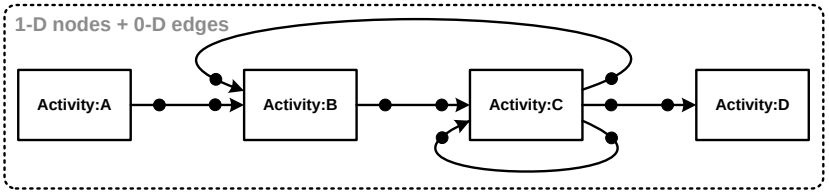
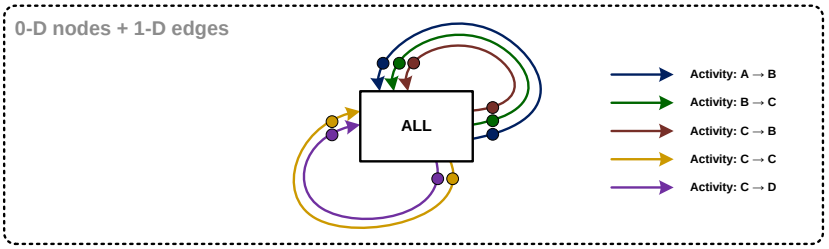


Figure 5.8: The only mC-net that can be generated from cuboid *()*.

- **1-D Cuboid** ***(Activity)*** represents the perspective characterized by the activities of the business process. Since this perspective is characterized by one dimension (dimension *Activity*), two possible multidimensional process models can be generated from this cuboid: one with 1-D nodes and 0-edges (Figure 5.9a) and another with 0-D nodes and 1-edges (Figure 5.9b).



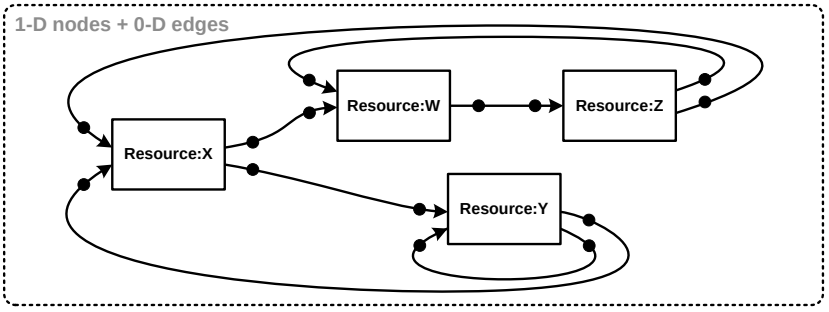
(a) Dimension *Activity* as event constraint.



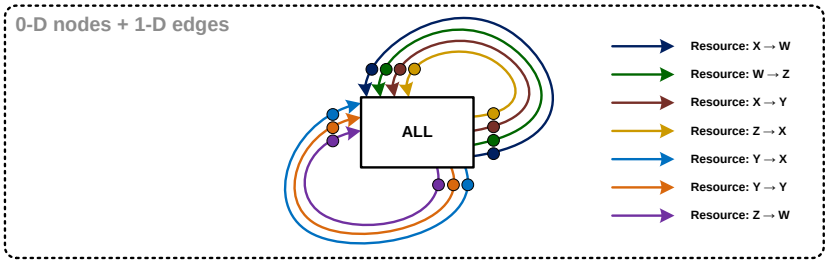
(b) Dimension *Activity* as workflow constraint.

Figure 5.9: The mC-nets that can be generated from cuboid *(Activity)*.

- **1-D Cuboid *(Resource)*** represents the perspective characterized by the resources of the business process. Since this perspective is characterized by one dimension (dimension *Resource*), two possible multidimensional process models can be generated from this cuboid: one with 1-D nodes and 0-edges (Figure 5.10a) and another with 0-D nodes and 1-edges (Figure 5.10b).



(a) Dimension *Resource* as event constraint.



(b) Dimension *Resource* as workflow constraint.

Figure 5.10: The mC-nets that can be generated from cuboid *(Resource)*.

- **2-D Cuboid *(Activity,Resource)*** represents the perspective characterized by the activities of the business process as well as the resources that who/which performed the activities. Since this perspective is characterized by two dimension (dimensions *Activity* and *Resource*), four possible multidimensional process models can be generated from this cuboid:
  - one process model with 2-D nodes and 0-edges (Figure 5.11a),
  - two process models with 1-D nodes and 1-edges (figures 5.11b and 5.11c), and
  - one process model with 0-D nodes and 2-edges (Figure 5.11d).

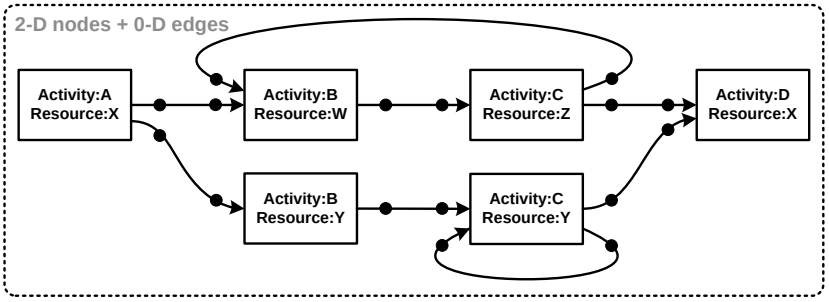
5.3.2 Business Process Querying

Process discovery techniques provide insight into the business process by answering different types of queries. A possible characterization of such types is the following:

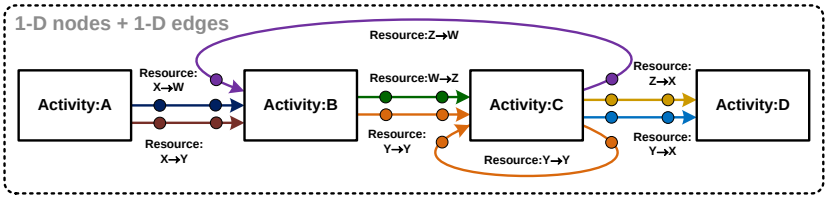
- **What** queries aim at the identification of events in the process by answering the following question: *What are the events in the process?* This is the most common type of query in process discovery.
- **Who** queries provide insight into the originators of the events. *Who are the event executors?* is the question that represents these queries. Social and organizational analyses are typical applications of this type of query.
- **When** queries identify the temporal references of the events. *When do the events occur?* is an example of the time-based questions that can be used to find temporal patterns or seasonal trends.
- **Which** queries link event-related information with the events or process instances. *Which are the objects associated with the events (or the process instances)?* is the generic question that defines this type of query. Traditionally, these queries are not explicitly supported in process discovery techniques.
- **How** queries describe the execution of events in a process by answering the following question: *How do the events are executed?* Basically, these queries provide insight into the process behavior, the main aspect of the business process analyses. Control-flow mining is an example of a process discovery application that supports this type of query.
- **Why** queries explain the decisions in the execution of a process. *Why do different process instances have different behavior?* is the question that represents this type of query. Decision point analysis is a typical application of these queries.

Designated *event-related queries*, *what*, *who*, and *when* queries identify aspects of process events. Designated *workflow-related queries*, *how* and *why* queries provide insight into the process behavior (workflow). Being both event- and workflow-queries, *which* queries focus on either process events or process behavior.

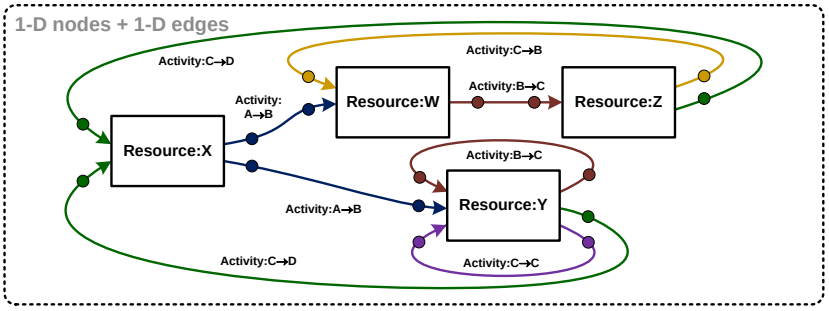
The process analysis is achieved by combining event- and workflow-queries. For example, a traditional process model combines *what* and *how* queries once that it identifies the activities in the process as well as the process behavior. An overview about the query orientation of common process mining techniques for process discovery and performance analysis is presented in Table 5.7. All of the techniques considered in this overview can be found in ProM [123].



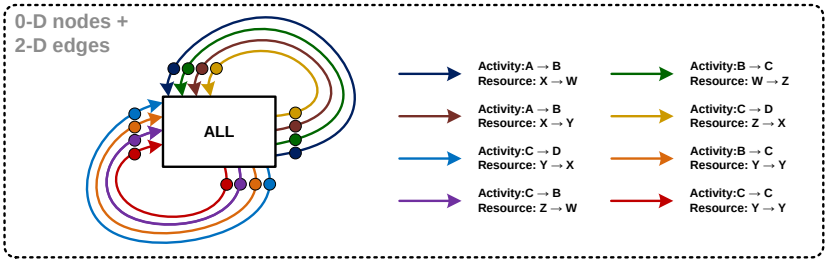
(a) Dimensions *Activity* and *Resource* as event constraints.



(b) Dimension *Activity* as event constraint and dimension *Resource* as workflow constraint.



(c) Dimension *Activity* as workflow constraint and dimension *Resource* as event constraint.



(d) Dimensions *Activity* and *Resource* as workflow constraints.

Figure 5.11: The mC-nets that can be generated from cuboid *(Activity,Resource)*.

	Events				Workflow	
Technique	*What*	*Who*	*When*	*Which*	*How*	*Why*
Control-flow mining	✓				✓	
Decision point analysis	✓					✓
Social network analysis	✓	✓				
Temporal analysis	✓	✓	✓		✓	
Sequence analysis	✓	✓	✓		✓	
Linear Temporal Logic (LTL)	✓	✓	✓			
Performance analysis	✓	✓	✓	✓		
Multidimensional process discovery	✓	✓	✓	✓	✓	✓

Table 5.7: Query orientation for different process mining techniques.

Event-related queries can be considered the simplest type of query once that their execution consists basically of information retrieval on process data. On the other hand, the execution of workflow-based queries depends on information about process behavior. Hence, it is necessary to apply some control-flow mining algorithm before executing the query.

Due to its multidimensional nature, the multidimensional process discovery approach covers a broader range of queries than any other process mining technique. This means that different aspects of the business process can be exploited in an integrated multidimensional analysis. Currently, similar analysis can be eventually achieved using traditional techniques after filtering the dimensions. However, applying a filter for all dimensions values is almost unfeasible since it would be extremely time consuming and the different results for the different values would not be integrated.

5.3.3 Cube Exploitation

As long as the necessary aspects of the business process are described in the data, business process querying can be performed in Event Cubes. As mentioned before, the process analysis is achieved by combining event- and workflow-queries. Actually, each different combination defines a process perspective, which is represented in the Event Cube by cuboids. Therefore, the cube exploitation consists of business process querying.

The result of a business process query is represented by a multidimensional process model, which can be considered as a multidimensional view of data. As in traditional data cubes, multidimensional views in Event Cubes can be defined interactively through some operations over the lattice of cuboids. Designated *discovery operations*, these operations facilitate the combination of event- and workflow-queries by adjusting the aspects of the business process in analysis. Since events and workflow are represented by different objects, discovery operations are extended to operate in different multidimensional spaces.

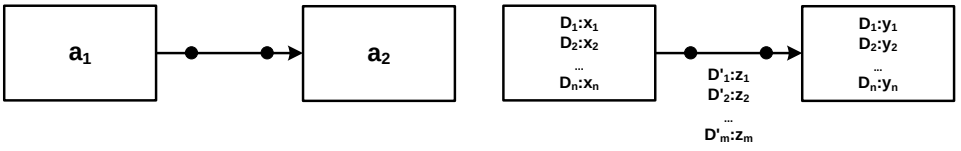
Multidimensional Spaces

A space can be defined as a set of objects. Considering that a process is represented by two types of objects (i.e., nodes and edges, using the notation introduced in Section 3.2), the space defined by process models consists of two subspaces.

Nodes represent the subspace of process events. In a traditional process model, a set of activities defines the nodes. Since an activity is an one-dimensional object, traditional process models consist of one-dimensional nodes. In a multidimensional process model, nodes are defined by a set of event occurrences. Since an event occurrence is an n-dimensional object, multidimensional process models consists of n-dimensional nodes.

Edges (or arcs) represent the subspace of transitions between process events. In a traditional process model, a set of – unconstrained – transitions defines the edges. Since an unconstrained transition is a zero-dimensional object (i.e., the transition is not characterized by any information), traditional process models consists of zero-dimensional edges. In a multidimensional process model, edges are defined by a set of constrained transitions. Since a constrained transition is an n-dimensional object, multidimensional process models consists of n-dimensional edges.

Figure 5.12 illustrates the dimensionality of nodes and edges in traditional and multidimensional process models. A traditional process model consists of one-dimensional nodes and zero-dimensional edges (Figure 5.12a). Therefore, a traditional process model can be designated as a one-dimensional model. A multidimensional process model consists of n-dimensional nodes and m-dimensional edges (Figure 5.12b). Consequently, a multidimensional process model can be designated as a $(n+m)$-dimensional model.



(a) Traditional process models. **(b)** Multidimensional process models.

Figure 5.12: Spaces of process models.

Discovery Operators

Like in traditional data cubes, the information of Event Cubes can be accessed by performing some OLAP-based operations over the lattice of cuboids. OLAP operations such as drill-down, roll-up, and pivot can be used to create multidimensional views of data by retrieving information from specific cuboids. The same operations can be used in the Event Cube for multidimensional process analysis.

Drill-Down adds a dimension to the multidimensional process model. This means that the process will be represented with more details (i.e., in a lower abstraction level). The drill-down operation can be performed over either nodes or edges. Figure 5.13 shows an example where a drill-down operation is performed over nodes. In this case, details regarding the given dimension are added to the nodes. Edges are represented in the same abstraction level.

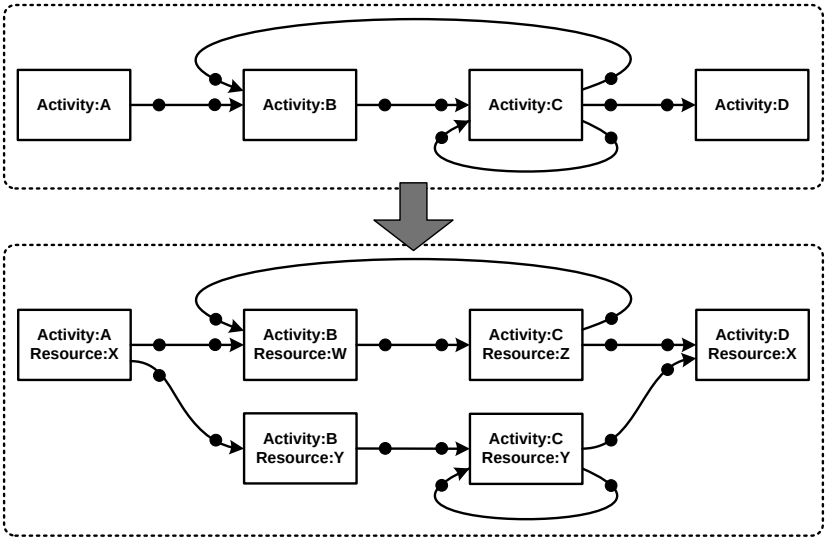


Figure 5.13: Drill-down by adding the dimension *Resource* to nodes.

Figure 5.14 presents an example where a drill-down operation is performed over edges. In this case, details regarding the given dimension are added to the edges. Nodes are represented in the same abstraction level.

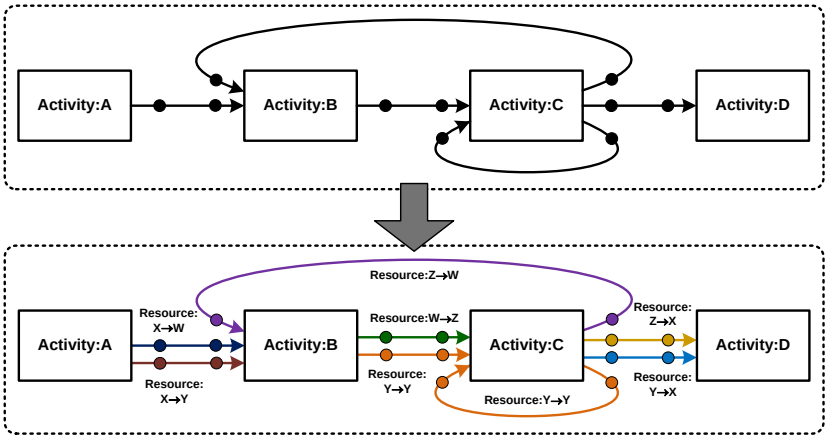


Figure 5.14: Drill-down by adding the dimension *Resource* to edges.

Roll-Up removes a dimension from the multidimensional process model. This means that the process will be represented with less details (i.e., in a higher abstraction level). The roll-up operation can be performed over either nodes or edges. Figure 5.15 shows an example where a roll-up operation is performed over nodes. In this case, details regarding the given dimension are removed from the nodes. Edges are represented in the same abstraction level.

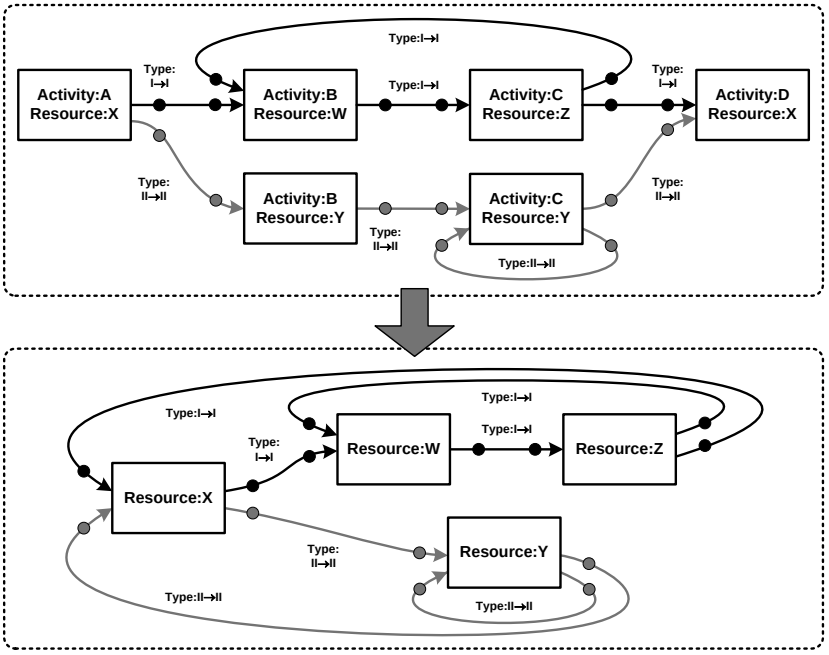


Figure 5.15: Roll-up by removing the dimension *Activity* from nodes.

Figure 5.16 presents an example where a roll-up operation is performed over edges. In this case, details regarding the given dimension are removed from the edges. Nodes are represented in the same abstraction level.

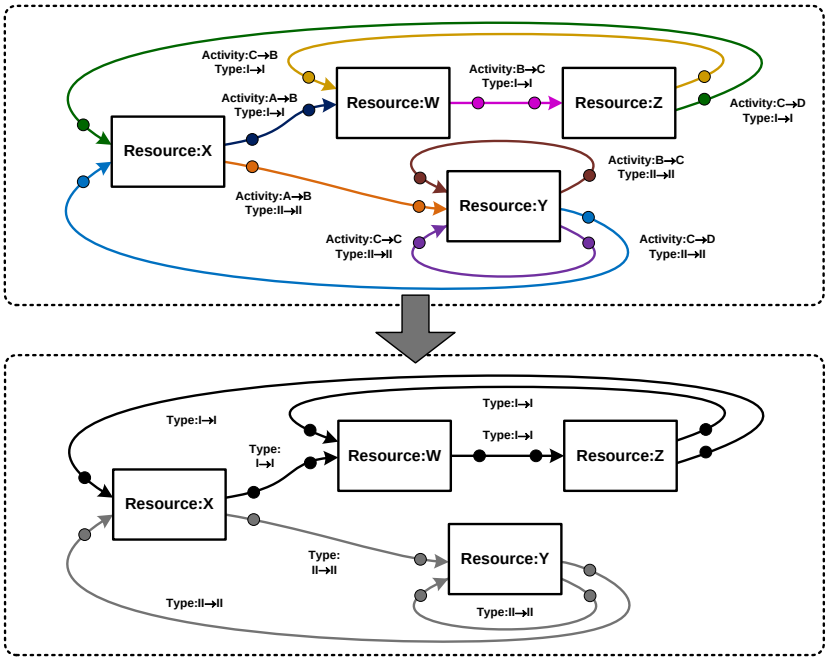


Figure 5.16: Roll-up by removing the dimension *Activity* from edges.

Pivot permutes the dimensions of the nodes and edges. Therefore, the perspective remains the same but the process information is presented in a different configuration. Figure 5.17 illustrates a pivot operation. In this case, the dimension in the nodes subspace (*Activity*) is swapped with the dimension in the edges subspace (*Resources*). The abstraction level of both nodes and edges remain the same.

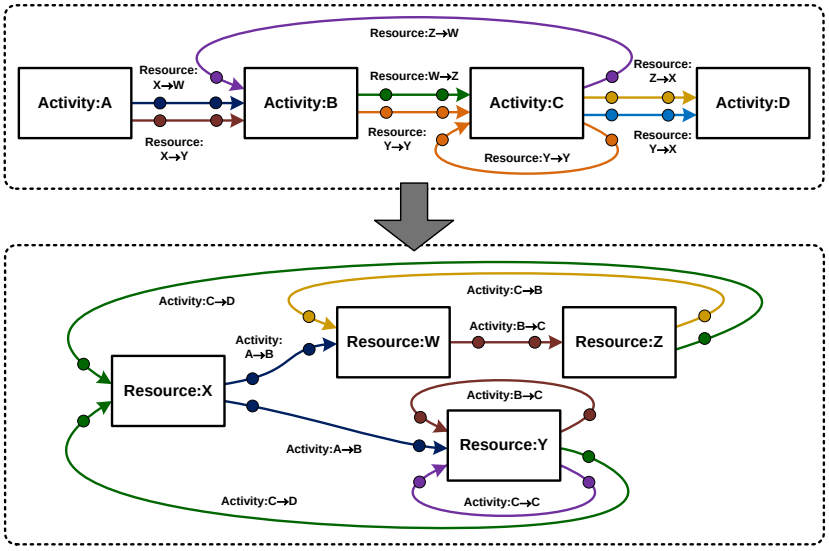
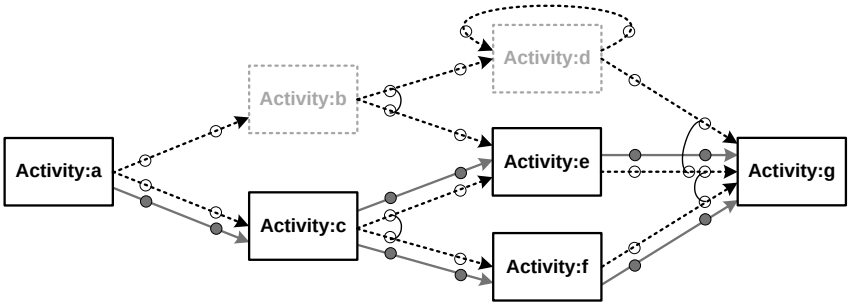


Figure 5.17: Pivot by permuting the dimensions of the nodes and edges.

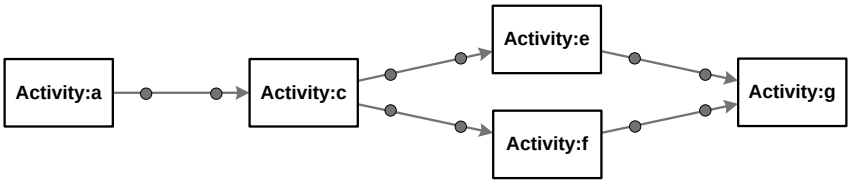
5.3.4 Filtering Process Behavior

As in traditional data cubes, multidimensional views in Event Cubes can be constrained interactively through some operations over the cuboids' information. Designated *filtering operations*, these operations facilitate the analysis of business process queries by constraining the process behavior in analysis. Like discovery operations, filtering operations can operate over both nodes and edges multidimensional subspaces. Furthermore, these operations can be performed by applying filtering conditions over either characterizing or measuring information.

Filtering edges consists of omitting (removing) specific dependency relations from a multidimensional process model. The dependency relations to be filtered out are determined by a filtering condition using workflow constraints. Whenever a dependency relation does not satisfy the filtering condition the edge representing the relation should be removed from the process model. Remark that filtering out a dependency relation may require the recomputation of the input and output bindings the relation is part of. Additionally, if the input and output bindings of a specific event occurrence consist only of empty bindings (after binding recomputation), then that occurrence should be removed from the process model. An example of filtering edges from a multidimensional process model is presented in Figure 5.18.



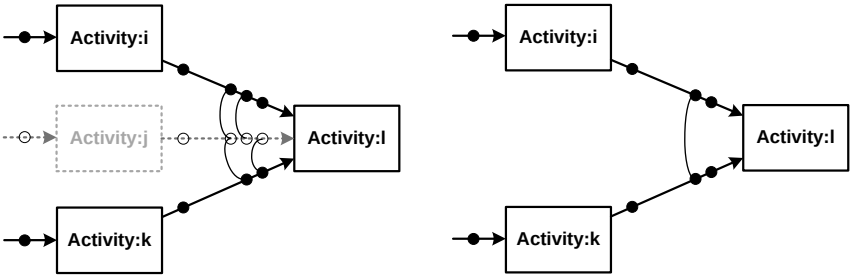
(a) The dependency relations (and event occurrences) to be omitted.



(b) The resulting multidimensional process model after filtering.

Figure 5.18: An example of filtering edges from a multidimensional process model.

Figure 5.18a identifies the dependency relations to be filtered out from the process model as well as the event occurrences that will not have any non-empty input and output bindings. These dependency relations and event occurrences are represented by dashed lines. Figure 5.18b shows the result of the filtering operation. Remark that, in this example, it was not necessary to recompute any input or output binding. Another example is presented in Figure 5.19 to illustrate the recomputation of bindings.



(a) The dependency relations (and event occurrences) to be omitted. **(b)** The resulting submodel after filtering.

Figure 5.19: An example of recomputing an input binding.

Figure 5.19a identifies the dependency relations to be filtered out from the process model as well as the event occurrences that will not have any non-empty input and output bindings. Once again, these dependency relations and event

occurrences are represented by dashed lines. Before filtering, the input bindings of $\{Activity{:}l\}$ are:

1. $\{\ (\{Activity{:}i\}, \bullet)\ \}$,
2. $\{\ (\{Activity{:}j\}, \bullet), (\{Activity{:}k\}, \bullet)\ \}$,
3. $\{\ (\{Activity{:}i\}, \bullet), (\{Activity{:}j\}, \bullet)\ \}$, and
4. $\{\ (\{Activity{:}i\}, \bullet), (\{Activity{:}j\}, \bullet), (\{Activity{:}k\}, \bullet)\ \}$,

with $\bullet$ and $\bullet$ representing the dependency relations' workflow constraints. By removing the binding $(\{Activity{:}j\}, \bullet)$ from the input bindings, the input bindings of $\{Activity{:}l\}$ after filtering are:

1. $\{\ (\{Activity{:}i\}, \bullet)\ \}$,
2. $\{\ (\{Activity{:}k\}, \bullet)\ \}$, and
3. $\{\ (\{Activity{:}i\}, \bullet), (\{Activity{:}k\}, \bullet)\ \}$.

Figure 5.19b shows the submodel as result of the filtering operation.

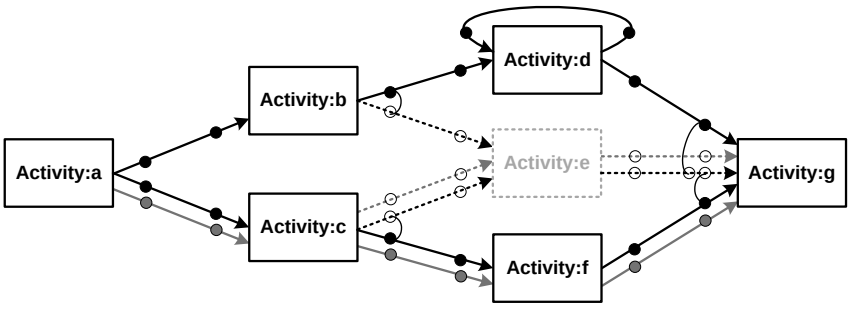
Filtering nodes consists of omitting (removing) specific event occurrences from a multidimensional process model. The event occurrences to be filtered out are determined by a filtering condition composed by event constraints. Whenever an event occurrence does not satisfy the filtering condition not only the node representing the occurrence should be removed from the process model but also every adjacent edge. Remark that, in order to keep the consistency of the process behavior, filtering out an event occurrence may require the insertion of *artificial dependency relations* in the process model. This can be done by replacing the event occurrence bindings by artificial dependency relations, linking these to the binding's activated inputs and outputs. An example of filtering a node from a multidimensional process model is presented in Figure 5.20.

Definition 5.1 (Artificial Dependency Relation) *An artificial dependency relation is defined as a relation that is not supported in the process data but exists in the process model to describe the process behavior of some filtered process events.*

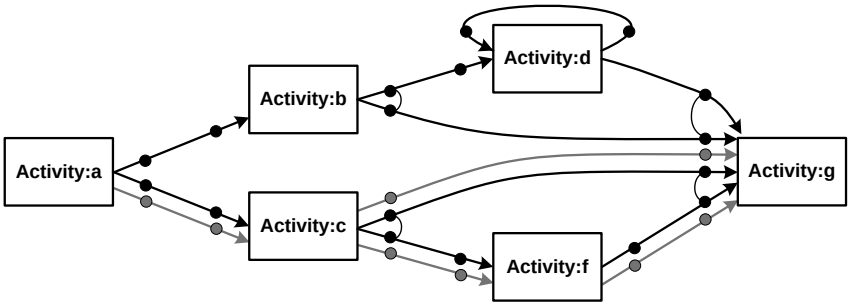
Figure 5.20a identifies the event occurrence to be filtered out from the process model as well as the corresponding adjacent dependency relations. These event occurrences and dependency relations are represented by dashed lines. Figure 5.20b shows the result of the filtering operation. Remark that three artificial dependency relations are introduced to replace the event occurrence bindings of $\{Activity{:}E\}$.

The generation of artificial dependency relations is defined in Algorithm 7. Basically, this process updates the dependency graph of a mC-net by

- removing a specific event occurrence and its related dependency relations, and
- adding – artificial – dependency relations to replace the process behavior characterized by the event occurrence bindings of the removed event occurrence.



(a) The event occurrence (and adjacent dependency relations) to be omitted.



(b) The resulting multidimensional process model after filtering.

Figure 5.20: An example of filtering nodes from a multidimensional process model.

Let us use Figure 5.21 to illustrate the generation process of artificial dependency relations. The multidimensional process model to be filtered is depicted in Figure 5.21a. Figure 5.21b identifies the objects to be removed from the process model. From these objects, $\{Activity{:}j\}$ is the event occurrence, while the dependency relations are

- $(\{Activity{:}i\}, \{Activity{:}j\}, \{Resource{:}x \rightarrow y\})$,
- $(\{Activity{:}j\}, \{Activity{:}l\}, \{Resource{:}y \rightarrow z\})$, and
- $(\{Activity{:}j\}, \{Activity{:}m\}, \{Resource{:}y \rightarrow w\})$.

Not described in the process model, the event occurrence bindings of $\{Activity{:}j\}$ are $B_1 = (I_1, O_1)$ and $B_2 = (I_1, O_2)$, where

- $I_1 = \{\ (\{Activity{:}i\}, \{Resource{:}x \rightarrow y\})\ \}$,
- $O_1 = \{\ (\{Activity{:}l\}, \{Resource{:}y \rightarrow z\})\ \}$, and
- $O_2 = \{\ (\{Activity{:}m\}, \{Resource{:}y \rightarrow w\})\ \}$.

Figure 5.21c presents the multidimensional dependency graph after filtering. The identified objects in Figure 5.21b are replaced by two artificial dependency relations. Remark that the dependency relations to be removed from the model are not given as argument but determined by event occurrence bindings (lines 3-8 in Algorithm 7). For example, the dependency relation

Algorithm 7: Generating Artificial Dependency Relations

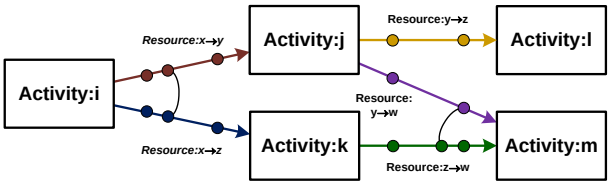
Input : The multidimensional dependency graph (DG) to be filtered, and the event occurrence X and its event occurrence bindings B.

Output: The filtered multidimensional dependency graph.

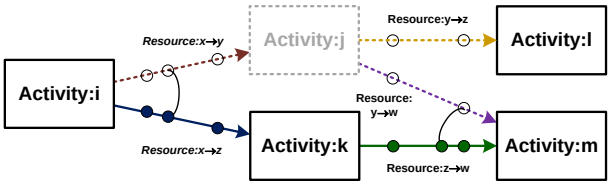
Method

```
1  R⁻ ← ∅, R⁺ ← ∅;
2  foreach event occurrence binding b ∈ B do
3      I ← the input binding of b;
4      foreach activated input (e_i, w_i) ∈ I do
5          R⁻ ← R⁻ ∪ {(e_i, X, w_i)};
       end
6      O ← the output binding of b;
7      foreach activated output (e_o, w_o) ∈ O do
8          R⁻ ← R⁻ ∪ {(X, e_o, w_o)};
       end
9      foreach activated input (e_i, w_i) ∈ I do
10         foreach activated output (e_o, w_o) ∈ O do
11             W ← ∅;
12             foreach workflow constraint (D:d_c → d_d) ∈ w_o do
13                 (D:d_a → d_b) ← the workflow constraint for dimension D in w_i;
14                 W ← W ∪ {D:d_a → d_d};
               end
15             R⁺ ← R⁺ ∪ {(e_i, e_o, W)};
           end
       end
   end
16 return DG ∪ R⁺ \ R⁻;
```

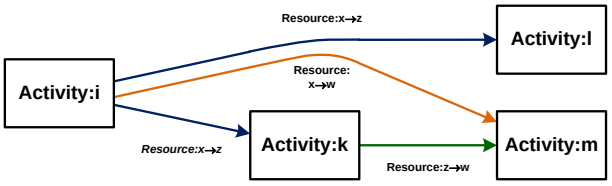
$(\{Activity{:}j\}, \{Activity{:}m\}, \{Resource{:}y \rightarrow w\})$ is determined by O_2, the output binding of B_2. The artificial dependency relations are also determined by event occurrence bindings (lines 9-15 in Algorithm 7). For example, the artificial dependency relation $(\{Activity{:}i\}, \{Activity{:}m\}, \{Resource{:}x \rightarrow w\})$ (represented by the orange edge) is determined by I_1 and O_2, the input and output bindings of B_2. When the multidimensional dependency graph is updated (i.e., after applying Algorithm 7), the event occurrence bindings need to be recomputed over the new dependency graph. This can be done using Algorithm 5. Finally, the multidimensional process model after filtering is depicted in Figure 5.21d.



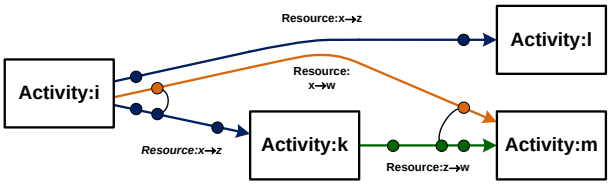
(a) The mC-net to be filtered.



(b) The event occurrence (and adjacent dependency relations) to be omitted.



(c) The multidimensional dependency graph after filtering.



(d) The mC-net after filtering.

Figure 5.21: An example of recomputing the event occurrence bindings.

Filtering Operators

OLAP operations such as slice-and-dice and top-k selection can be used to constraint the multidimensional views of data by applying some filtering conditions. The same operations can be used in the Event Cube for filtering process behavior in multidimensional process models. Remark that, unlike the discovery operators that are performed over the lattice of cuboids, filtering operators are applied on the process models generated from the cuboids.

Slice and Dice restricts the objects in the multidimensional process model by using filtering conditions over the dimension values that characterize the object. The objects that do not satisfy the filtering conditions are removed from the pro-

cess model. The perspective remains the same but less process information is presented. The slice and dice operation can be performed over either nodes or edges. Figure 5.22 shows an example where a slice operation is performed over edges. In this case, only one filtering condition ($Type = II \rightarrow II$) is applied.

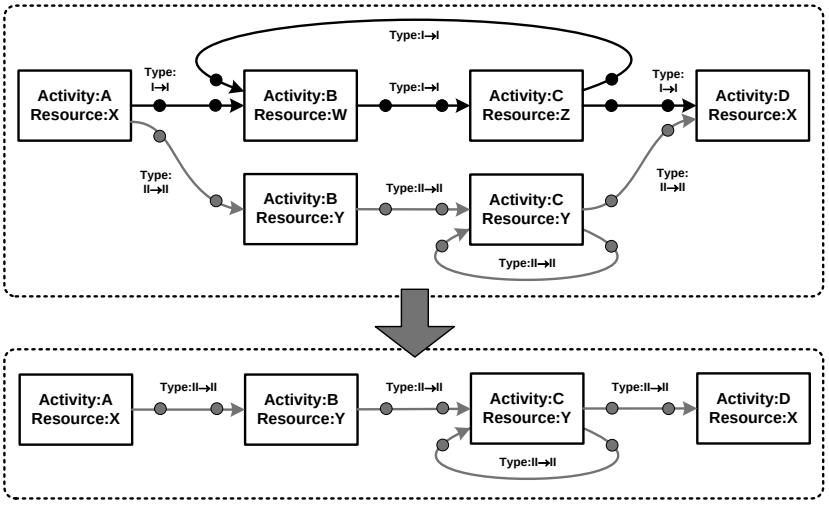


Figure 5.22: Slice and dice by using the filtering condition ($Type = II \rightarrow II$).

Figure 5.23 presents an example where a slice and dice operation is performed over edges and nodes. In this case, two filtering conditions ($Type = II \rightarrow II$ and $Resource = Y$) are applied.

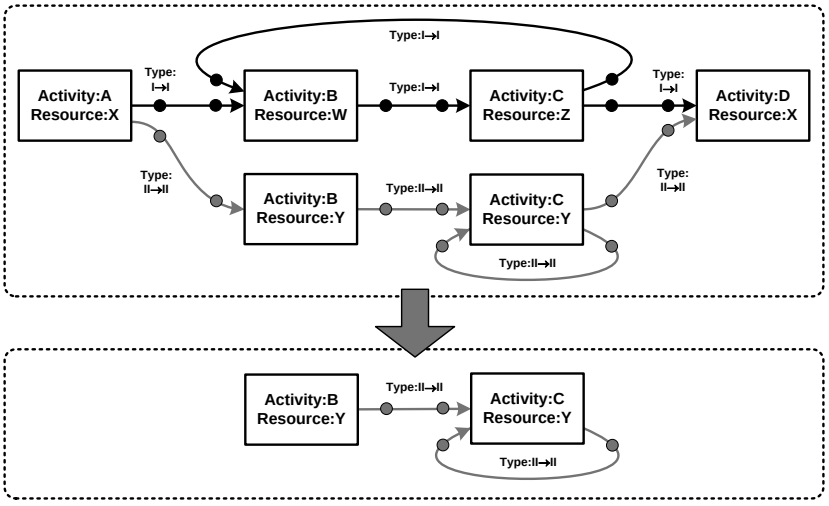


Figure 5.23: Slide and dice by using the filtering conditions ($Type = II \rightarrow II$ and $Resource = Y$).

Top-k Selection restricts the objects in the multidimensional process model by ranking the objects with respect to some given measure that summarize the object.

Eventually, filtering conditions over the measure may be used to restrict the ranking. Only the first k objects in the ranking are considered in the process model. The perspective remains the same but less process information is presented. The top-k selection operation can be performed over either nodes or edges. Figure 5.24 shows an example where a top-k selection is performed over event occurrences. In this case, the process model is built considering only the top-5 most frequent event occurrences (i.e., every node representing an event occurrence outside the top-5 ranking is filtered out).

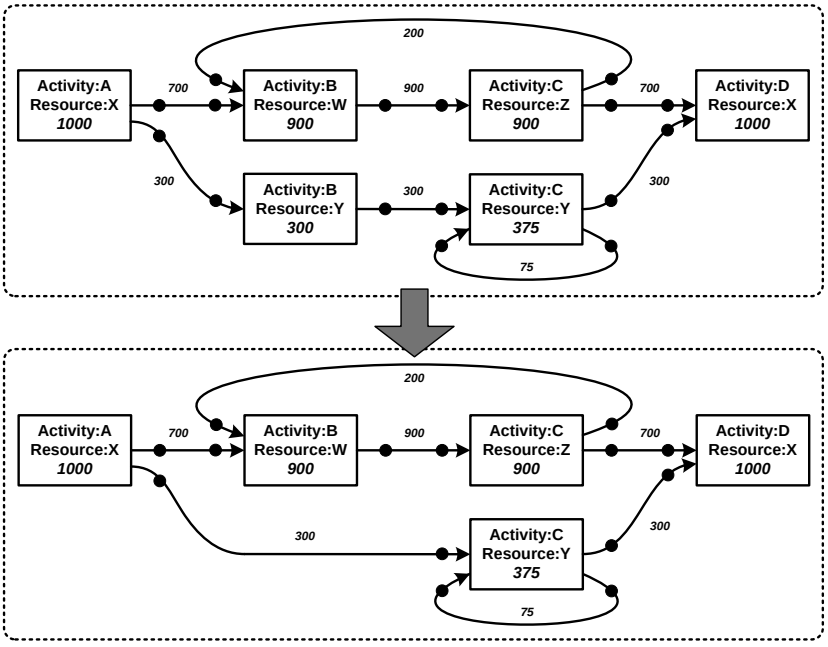


Figure 5.24: Top-k selection by considering the top-5 most frequent event occurrences.

Figure 5.25 presents an example where a top-k selection is performed over dependency relations. In this case, the process model is built considering only the top-5 most frequent dependency relations (i.e., every edge representing a dependency relation outside the top-5 ranking is filtered out). Remark that, no dependency relation (or event occurrence) with an equal measure value as another case within the top-k is discarded. This explains the fact that there are 6 dependency relations in the resulting model of Figure 5.25.

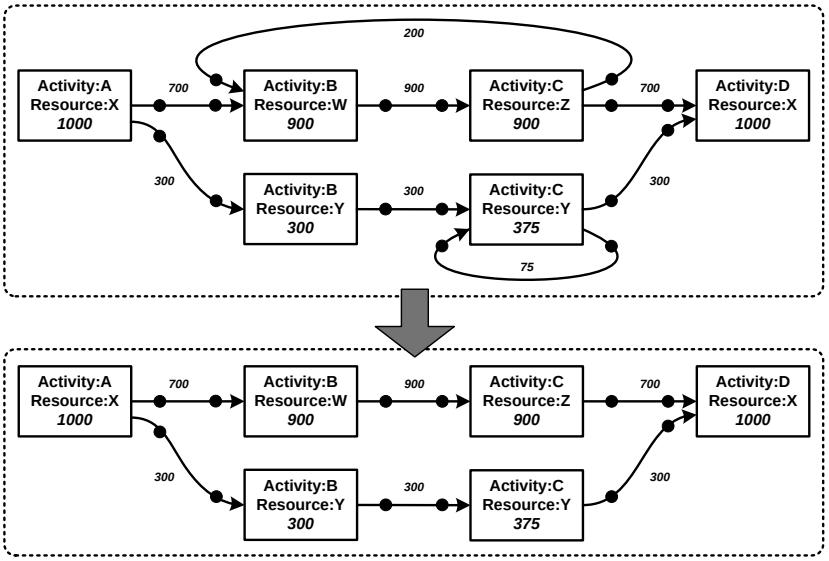


Figure 5.25: Top-k selection by considering the top-5 most frequent dependency relations.

5.4 Challenges

In this section, we discuss some challenges and issues related to the discovery and analysis of business processes using Event Cubes.

5.4.1 Curse of Dimensionality

Multidimensional process models are defined in Subsection 3.2.2. These models are a generalization of traditional process models in which event occurrences (i.e., group of process events) as well as their dependency relations can be constrained by multiple dimensions. Depending on the cardinality of each dimension considered in the model, the number of event occurrences and/or dependency relations may be significant enough to turn the process model unreadable. Known as *curse of dimensionality*, this issue is even more evident if the process behavior is complex and unpredictable. Figure 5.26 depicts a well-known case of curse of dimensionality in traditional process models, a *spaghetti model*.

Spaghetti models are process models characterized by a high number of dependency relations (and, eventually, activities). Typically representing low-structured processes, these process models are difficult to analyze once that the process behavior cannot be clearly characterized by representative sequences of activities. This is a consequence of the high cardinality of activity relationships (i.e., the pairs of activities that define the dependency relations).

Spaghetti models can also exist as multidimensional process models. Figure 5.27 shows a multidimensional process model that can be considered as a – multidimensional – spaghetti model. In this case, the high number of dependency relations is explained mainly by the cardinality of the dimensions used to characterize the relations.

One possible solution to minimize the impact of the dimensionality is the usage of

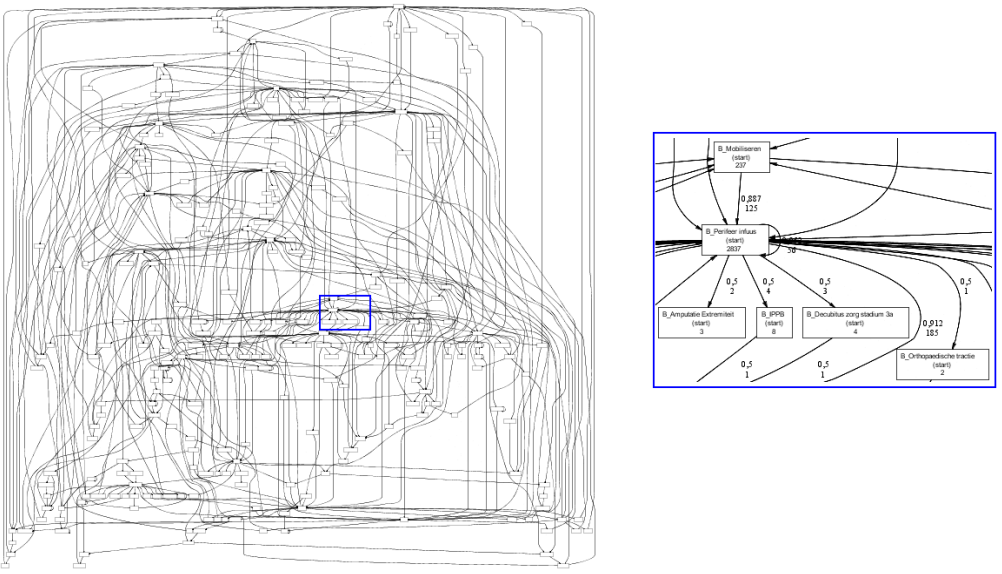


Figure 5.26: An example of a traditional spaghetti model.

simplified model representations such as the aC-net notation. Simply using two different kinds of elements to represent aC-nets (nodes and edges), it is possible to provide a clear picture of the process without omitting information. Complex relationships between activities (i.e., the splits and joins patterns) are provided separately. This representation has proven to be effective specially in low-structured processes [130].

Another strategy to handle the curse of dimensionality issue is the application of filtering operations on the multidimensional process models. Introduced in Subsection 5.3.4, filtering operations can decrease the number of objects in the process model by omitting nodes or edges according to a given criteria. For obvious reasons, filtering is an effective solution for the curse of dimensionality, but it holds two main disadvantages. First, it may not be easy to define a good filtering criteria, which may lead to misadjusted results. Second, in order to ensure process behavior consistency, some filtering operations generate the so-called artificial dependency relations. Since these dependency relations do not occur explicitly in the process data, it is not possible to compute any measuring information about artificial dependency relations. Consequently, some of the process behavior cannot be evaluated with regard to some measure.

An alternative solution – to the ones already considered in this thesis – is proposed in the Fuzzy Miner approach [47]. Basically, this solution consists of presenting the process model through multiple abstraction layers. The lowest abstraction layer represents business process by considering all existing activities and dependency relations (i.e., all the process behavior). As the abstraction level increases, simpler views of the business process are provided by

- preserving the process behavior considered significant,
- aggregating closely related elements of the process behavior considered insignificant as individuals but significant as a group, and
- omitting unrelatable elements of the process behavior considered insignificant.

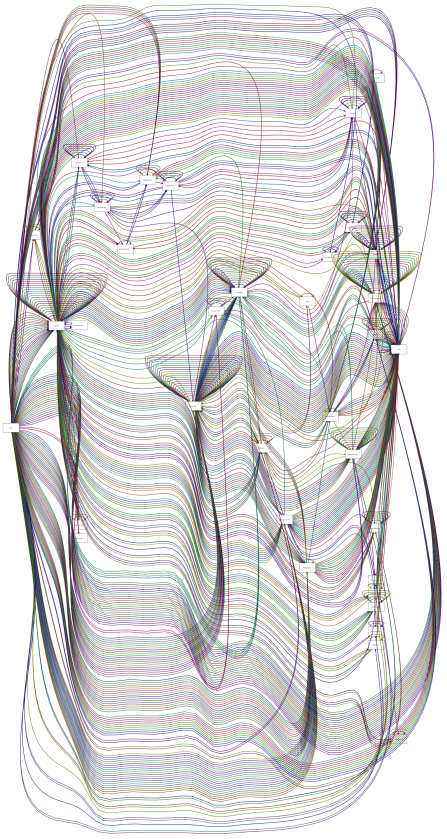


Figure 5.27: An example of a multidimensional spaghetti model.

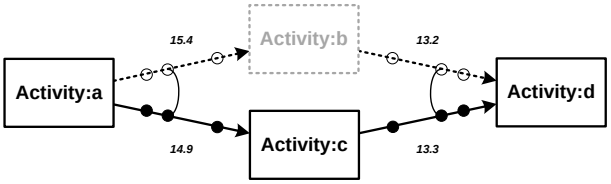
Simplified process models (i.e., the Fuzzy models introduced in [47]) can indeed be an effective solution for the curse of dimensionality issue. However, this approach is based on a process notation in which some elements of the process behavior cannot be characterized (e.g., splits and joins).

5.4.2 Measuring Artificial Dependency Relations

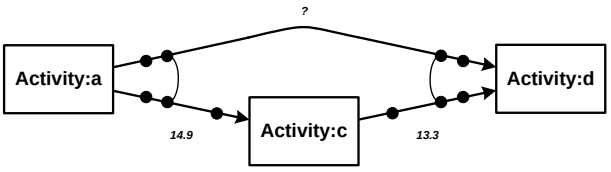
Filtering operations are described in Subsection 5.3.4. These operations can be performed over either event occurrences or dependency relations. Filtering dependency relations simply consists of removing – from the model – relations that do not satisfy some specific filtering condition as well as event occurrences that become inactive without the filtered dependency relations. Being more complex than filtering dependency relations, filtering event occurrences consists of removing – from the model – occurrences that do not satisfy some specific filtering condition as well as their dependency relations. This means that, for ensuring that the original process behavior is represented in the filtered process model, artificial dependency relations need to be added to the process model. This can be achieved by replacing the event occurrence bindings, but there still are some unsolved issues.

Figure 5.28 presents a multidimensional process model from which an event occur-

rence is being removed through a filtering operation. Figure 5.28a identifies both event occurrence and dependency relations to be removed from the model. Figure 5.28b shows the resulting multidimensional process model after filtering. In both process models, the numerical information describe the average waiting time between event occurrences.



(a) The event occurrence (and adjacent dependency relations) to be omitted.



(b) The resulting multidimensional process model after filtering.

Figure 5.28: Another example of filtering nodes from a multidimensional process model.

Generating an artificial dependency relation is based on event occurrence bindings. For example, the artificial relation in the resulting process model is defined by the only event occurrence binding of $\{Activity{:}b\}$:

$$\big(\ \{\ (\{Activity : a\}, \bullet)\ \}\ ,\ \{\ (\{Activity : d\}, \bullet)\ \}\ \big),$$

with $\bullet$ representing a specific workflow constraint that characterize the dependency relations. By using this binding, it is possible to determine that the process behavior described by $\{Activity{:}b\}$ can be replaced by an artificial dependency relation connecting $\{Activity{:}a\}$ to $\{Activity{:}d\}$ and constrained by the workflow constraint represented by $\bullet$. However, since artificial dependency relations are not described explicitly in the process data, the measuring information (in this case, the average waiting time) of this new artificial relation is not materialized in the Event Cube. Therefore, no measuring information can be provided for artificial dependency relations.

A potential solution for this issue relies on computing on-the-fly the measuring information of artificial dependency relations. This means that, after filtering a multidimensional process model, the missing information can be computed using the inverted index. For most measures, the measuring information can be derived directly using the sets of event identifiers of the artificial dependency relation. Using the inverted index, these sets can be easily computed through the event and workflow constraints that characterize the dependency relation. For measures that rely on the positioning of process events (e.g., the control-flow measure *Direct Successor*), the sets of event identifiers must be first normalized in order to ensure the correct positioning of process events taking into account the filtered cases.

Computing on-the-fly of the measuring information of artificial dependency relations seems to be an effective solution. However, this solution will necessarily have a negative impact on the performance of the entire framework.

5.4.3 Parameter Selection

The Multidimensional Heuristics Miner is introduced in Subsection 3.3.2. Based on the well-known Heuristics Miner algorithm, the Multidimensional Heuristics Miner is a control-flow technique adapted to multidimensional process discovery. As any other Heuristics Miner-based technique, this technique relies on a set of thresholds as well as other parameters. All of these parameters allow the user to tune the process models according to specific interests or requirements.

As demonstrated in Section 5.3.1, several distinct multidimensional process models can be generated from an Event Cube. These process models can be built on-the-fly by applying the Multidimensional Heuristics Miner over the cuboids of the Event Cube, which represent the different perspectives of the same business process. This means that, for the different process models, there may be the need of parameter tuning in order to improve the quality of the process models. Even being possible to update on-the-fly the process models with the different parameter settings, the parameter selection may be a bottleneck in the discovery and analysis of business processes.

Parameter optimization is a potential solution for this issue. This approach has been successfully employed in a benchmark in which C-nets can be evaluated [80]. The idea is to assess the best parameter setting by generating and evaluating several process models. Following the cross-validation strategy, the model evaluation consists of the parsing of some process instances on a process model. In this approach, not only positive cases are considered but also artificial negative cases.[3] By counting the process instances and – inputs and outputs – bindings parsed correctly, it is possible to determine the quality of the process model according to a specific fitness measure. An overview of fitness measures is provided in [80].

There are some drawbacks in applying the parameter optimization in the Event Cube materialization process. The first is the need of implementing an effective method for generating negative cases. Even considering simple methods such as the random noise injection [132], the generation of negative cases can easily have a significant impact on the performance of the materialization process. The second main drawback is the need of assessing a great number of parameter combinations. Although the search space can be reduced by using heuristics, the parameter setup assessment also affects negatively the performance of the materialization process.

5.4.4 Process Model Consistency across Perspectives

By performing discovery operations on the Event Cube (Subsection 5.3.3), it is possible to exploit different perspectives of the business process. This exploitation consists of discovering the business process by focusing on specific dimensions of interest. This means that the behavior described in the process data can be analyzed from different levels of abstraction.

[3]Positive cases can be described as process instances that describe behavior that occurred in a business process. On the other hand, negative cases can be defined as process instances that describe behavior that cannot occur in a business process. Usually, process data do not contain negative cases.

The structure of an Event Cube consists of a lattice of cuboids representing the different process perspectives. Each cuboid organizes the necessary measurements (i.e., the control-flow measures introduced in Subsection 5.2.3) to build multidimensional process models according to specific dimensions of interest. By querying and retrieving the cuboid's information, it is possible to discover and analyze the business process from multiple perspectives. It is imperative that the results of these analyses are consistent.

The process model consistency across perspectives depends directly on the control-flow algorithm. Since each perspective consists of a partition of the process data, assuring that the control-flow algorithm generates consistent process models across perspectives is extremely difficult, if not impossible. This happens due to some reasons. First, different partitions may have distinct characteristics (e.g., data quality and quantity). Second, the considered control-flow algorithm (i.e., the Multidimensional Heuristics Miner) is not able to mine very particular process behaviors. Third, the process discovery on the Event Cube follows an unsupervised approach (i.e., the results are not confirmed to be correct).

Figure 5.29 illustrates an example where a roll-up operation leads to inconsistency in the results. By removing the dimension *Activity* from nodes of the two-dimensional process model (i.e., the model at the top of the figure), it is expected that the one-dimensional process model over the dimension *Type* (i.e., the model at the bottom of the figure) represents the same process behavior. However, this does not happen in this example once that the Multidimensional Heuristics Miner is not able to mine a combination of length-one and -two loops such as the one that occurs in the process model at the bottom of the figure. As a result, the length-two loop cannot be mined (the dependency relations represented in the figure by the dashed lines).

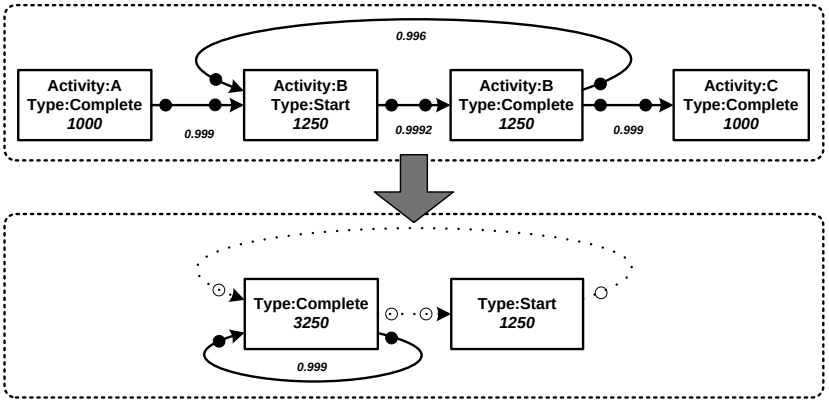


Figure 5.29: Example of inconsistency in the results after performing a roll-up operation.

In Figure 5.29, the process model inconsistency exists because the event occurrences $\{Type{:}Start\}$ and $\{Type{:}Complete\}$ have no dependency relation between them. Remark that the dependency relations represented by the dashed lines simply identifies the missing relations in the process model. Hence, the only dependency relations that are consistently represented in both process models are the one that goes from activities B to C and the length-two loop.

In order to get a better understanding of the root causes to the process model inconsistency in Figure 5.29, the mining of the dependency graph is described as follows.

Let $S = \{Type{:}Start\}$ and $C = \{Type{:}Complete\}$ be the event occurrences being analyzed. Assuming that there is no noise in the process data, the necessary dependency measures for mining the dependency graph are calculated as follows.

$$C \Rightarrow_W C = \frac{|C >_W C|}{|C >_W C| + 1} = \frac{1000}{1000 + 1} = 0.999$$

$$S \Rightarrow_W S = \frac{|S >_W S|}{|S >_W S| + 1} = \frac{0}{0 + 1} = 0$$

$$C \Rightarrow_W S = \frac{|C >_W S| - |S >_W C|}{|C >_W S| + |S >_W C| + 1} = \frac{1250 - 1250}{1250 + 1250 + 1} = \frac{0}{2501} = 0$$

$$S \Rightarrow_W C = C \Rightarrow_W S = 0$$

$$C \Rightarrow^2_W S = \frac{|C \gg_W S| + |S \gg_W C|}{|C \gg_W S| + |S \gg_W C| + 1} = \frac{1250 + 1250}{1250 + 1250 + 1} = \frac{2500}{2501} = 0.9996$$

$$S \Rightarrow^2_W C = C \Rightarrow^2_W S = 0.9996$$

Assuming the thresholds default values (i.e., $\sigma_D = \sigma_{L1L} = \sigma_{L2L} = 0.9$ and $\sigma_R = 0.05$), the multidimensional dependency graph of the one-dimensional process model of Figure 5.29 is computed as follows. According to Algorithm 4, $E = \{S, C\}$ is the set of event occurrences (line 1). Since $C \Rightarrow_W C \geq 0.9$, $D_1 = \{(C, C)\}$ is the set of length-one loops (line 2). Although $C \Rightarrow^2_W S \geq 0.9$, the length-two loop CSC is not added to the set of length-two loops (D_2) because $(C, C) \in D_1$. Hence, $D_2 = \emptyset$ is the set of length-two loops (line 3). Since $S \Rightarrow_W C = C \Rightarrow_W S = 0$, S cannot be considered as cause or follower of C (and vice-versa). Consequently, $D_{in} = D_{out} = D''_{in} = D''_{out} = \emptyset$ (lines 4, 5, 10 and 11), i.e., no further dependency relations are added to the dependency graph. Finally, the final dependency graph is determined by union of D_1, D_2, D''_{in} and D''_{out} (line 12). Therefore, as illustrated in Figure 5.29, the length-one loop (C, C) is the only dependency relation in the dependency graph.

A potential solution for this issue relies on an alternative strategy for performing process discovery on the Event Cube. Instead of materializing the control-flow measures for all cuboids of the Event Cube, this is done only for the 1-D cuboid *(Activity)*. Given a parameter setting, a process model with 1-D nodes over dimension *Activity* and 0-D edges is generated from this cuboid. This model is then indexed and used as reference process model. Rather than considering the characteristics of the process behavior, the index only takes into account the identifiers that define the dependencies relations in the reference process model. Hence, given a pair of event occurrences (x,y), it is possible to determine whether there is a dependency relation from x to y. Being T_x and T_y the set of event identifiers of x and y, a dependency relation from x to y exists if the index contains at least one pair of identifiers (a,b), where $a \in T_x$ and $b \in T_y$. The dependency graph of any multidimensional process model can be built by checking the existence of dependency relations for every possible pair of event occurrences. The event occurrence bindings of the new process model can be computed using Algorithm 5. Further details about this solution are provided in Appendix A.

Building multidimensional process models from a reference process model seems to be an effective solution to ensure process model consistency in the Event Cube. However, applying this solution in the Event Cube requires not only the implementation

of an additional index but also the redesign of the Event Cube exploitation processes. Additionally, further research is still necessary to validate and evaluate the approach.

5.5 Summary

Four components of the multidimensional process discovery approach were discussed in this chapter.

- **ETL** (Extraction, Transformation, and Loading) is the process of extracting process data from their sources, transforming the data according to specific requirements (e.g., deriving process information as described in Subsection 5.2.2), and indexing the data in an inverted index.
- **Inverted index** is presented in Subsection 5.2.1 as an indexing technique for multidimensional datasets. This technique is introduced to facilitate the direct access to process data characterized by specific constraints.
- **Cube processor** is described in Subsection 5.2.4 as a materialization process of an Event Cube.
- **Event Cube** is introduced in Section 5.2 as a multidimensional data structure designed to organize summarizing information about different perspectives of a process described in some given process data. Summarizing information consists of control-flow measures and performance measures. Control-flow measures facilitate the execution on-the-fly of a control-flow algorithm (e.g., the Multidimensional Heuristics Miner) on a specific business perspective. Performance measures can be used to enhance the multidimensional process model with measuring information.

All components described above are implemented as ProM 6 plugins, or objects that can be executed in the ProM framework [123, 125].

Answering research questions q_3 and q_4, dynamic multidimensional process discovery and analysis (cf. Section 5.3) can be achieved by exploiting an Event Cube. This exploitation can provide insight into different perspectives of the business process. By integrating the *organization*, *time*, and *data* perspectives into the *control-flow* one, dynamic multidimensional process discovery and analysis can provide the same kind of knowledge as those process discovery perspectives, but in an integrated environment, on-the-fly, and at different levels of abstraction.

Chapter 6

Process Patterns

In this chapter, we present some possible non-trivial patterns that can be automatically extracted from a multidimensional process model. Designated as *process patterns*, these patterns can be used to gain further insight into the business process. Process patterns can be extracted by applying common data mining functionalities such as [40]:

Summarization: These functions describe the cases by providing a more compact description of their characteristics. For example, calculating the average value of a customer order.

Deviation Analysis: These functions detect significant changes in the data (e.g., outliers). For example, detecting fraud based on unusual call behaviors.

Rule Learning: These functions search for significant dependencies between variables. For example, determining which products are frequently bought together.

Classification: These functions map a case to one of several predefined classes. For example, the classification of an e-mail as legitimate or spam.

Clustering: These functions describe the cases by grouping them into a finite set of categories or clusters according to their similarity. For example, finding customer groups with similar behavior.

For further details and an overview of data mining applications see [50].

Figure 6.1 positions this chapter with respect to the multidimensional process discovery approach. In the next sections, we firstly identify the different types of patterns that can be extracted from a multidimensional process model. Then, we characterize the main data mining functionalities, discussing how those functionalities may be applied for extracting process patterns from a multidimensional process model. Finally, we introduce some possible data mining-based techniques for extracting process patterns. Remark that these techniques should be seen as examples of the application of common data mining functionalities. Other techniques not considered in this thesis may be used for extracting of process patterns as well.

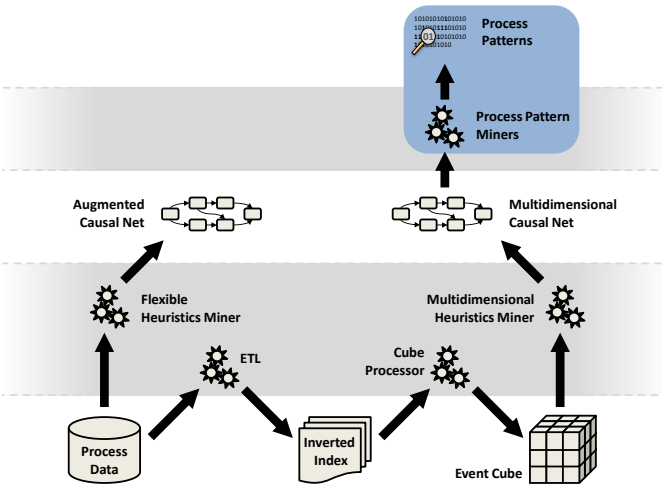


Figure 6.1: Positioning of Chapter 6 with respect to the multidimensional process discovery approach.

6.1 Pattern Characterization

A pattern derived from a multidimensional process model is called *process pattern*, and is defined by the following definition.

> *A process pattern consists of a set of process instances, event occurrences or dependency relations in which all the members are related to each other by a specific rule.*

There are three types of process patterns:

- **Event based-patterns** are process patterns formed by **event occurrences**. Focusing on the execution of process events, this type of pattern identifies non-trivial relationships among event occurrences such as *the set of process events that are frequently executed together*.
- **Flow based-patterns** are process patterns formed by **dependency relations** or **event occurrence bindings**. This type of process pattern focuses on the execution relationships of process events described in the process model, providing thus insight into how the events are executed. An example of a flow-based pattern is given by following rule: *for a specific product, a failed attempt of a simple repair leads always to the execution of a complex repair.*
- **Instance based-patterns** are process patterns formed by **process instances**. Rather than process events or their relationships, this type of process pattern focuses on the execution of process cases, from when the case starts (first process event) till it ends (last process event). Hence, instance-based patterns provide insight into the sequences of process events that define the process instances. An example of an instance-based pattern is given by the following rule: *the average throughput time of a specific case of repair process is 25% higher than the average throughput time of all cases of repair process.*

6.2 Pattern Extraction

The extraction of process patterns is model-driven. Hence, taking a multidimensional process model as reference, different types of process patterns (i.e., instance, event, and flow based-patterns) can be can be discovered by applying data mining functionalities. In this section, we describe how these functionalities can be used in the business process context. The main data mining functionalities are considered: (i) summarization, (ii) deviation analysis, (iii) rule learning, (iv) classification, and (v) clustering.

6.2.1 Summarization

The summarization is the most straightforward function to be applied on process data. By making use of the locations where process instances, event occurrences and dependency relations appear in the process data (i.e., the identifiers), it is possible to compute specific measures for analysis such as frequency or average throughput time. Considering the different multidimensional process model's objects, there are three kinds of summarization: *instance based-summarization*, *event based-summarization*, and *flow based-summarization*.

Instance based-summarization is defined to describe specific characteristics of process instances. This is achieved by computing instance-based measures on sets of process instance identifiers that locate where the process instances appear in the process data.

Event based-summarization is defined to describe specific characteristics of event occurrences. This is achieved by computing event-based measures on sets of event identifiers that locate where the event occurrences appear in the process data.

Flow based-summarization is defined to describe specific characteristics of dependency relations or event occurrence bindings. This is achieved by computing workflow-based measures on sets of event identifiers that locate where the dependency relations appear in the process data.

6.2.2 Deviation Analysis

Since process instances, event occurrences and dependency relations do not explicitly hold any numerical information, the deviation analysis needs to be combined with – numerical – summarization. By making use of the summarization outcome, this function computes a *deviation measure* (e.g., the standard score) that describes how much the case deviates from the average. Then, by applying threshold values, it is possible to identify – and eventually filter – the significantly deviated cases.

Considering the standard score (Z-Score) as deviation measure, the deviation analysis function can be defined as follows.

Definition 6.1 (Deviation Analysis Function) *Let X be a set of multidimensional process model's objects, $x \in X$ the object in analysis, and $f^{sum}(x)$ the summarization function for x. The deviation analysis function is defined as follows*

$$f^{dev}(x) = \frac{f^{sum}(x) - \mu}{\sigma}, \tag{6.1}$$

where μ and σ are the arithmetic mean and standard deviation of all objects in X.

The object x is considered significantly deviated (or outlier) if $|f^{dev}(x)| \geq \tau_{dev}$, with τ_{dev} being a given threshold value.

A potential application of the deviation analysis function is the identification of process instances (or process events) that deviate significantly from the average in terms of a given measure of interest (e.g., throughput time or execution cost). Table 6.1 shows an example in which process instances are evaluated with respect to their standard scores for a generic measure of interest. All process instances with standard score absolute value greater than 3.0 (i.e., $|f^{dev}(i_x)| \geq 3.0$) are considered as outliers.

Process Instance i_x	*Measure* $f^{sum}(i_x)$	*Standard Score* $f^{dev}(i_x)$
i_1	150	1.3
i_2	130	0.0
i_3	125	-0.3
...	...	...
i_n	5	**-8.3**
	$\mu = 130.0$	$\tau_{dev} = 3.0$
	$\sigma = 15.0$	

Table 6.1: Identifying significantly deviated process instances.

6.2.3 Rule Learning

Unlike the deviation analysis, the rule learning does not necessarily rely on summarization. Using the process instances as reference, the rule learning function determines non-trivial relationships between process instances, event occurrences, or dependency relations. These relationships are expressed as rules under different forms. Frequent itemsets (groups of event occurrences or dependency relations) and subsequences, association rules, and gradient relationships are common forms of rules produced by this function.

Frequent itemsets and subsequences

Frequent itemsets are process patterns that express that a group of event occurrences or dependency relations appear frequently together in a process instance. Similarly, frequent subsequences are process patterns that focus on sequences of event occurrences or dependency relations instead of individual objects. As an example, let us consider the product repair process depicted by the multidimensional process model in Figure 6.2. The set of event occurrences $\{Register, Analyze, Test, Close\}$ is a possible frequent itemset. The sequence of dependency relations *Register* $\rightarrow$ *Analyse* $\rightarrow$ *Simple Repair* $\rightarrow$ *Test* (for the solid line dependencies) is a possible frequent subsequence.

Definition 6.2 (Frequent Itemset (or Subsequence)) *An itemset (or subsequence) I is* frequent *if the support of I is greater or equal than a given minimum support threshold τ_{sup} (i.e., $support(I) \geq \tau_{sup}$).*

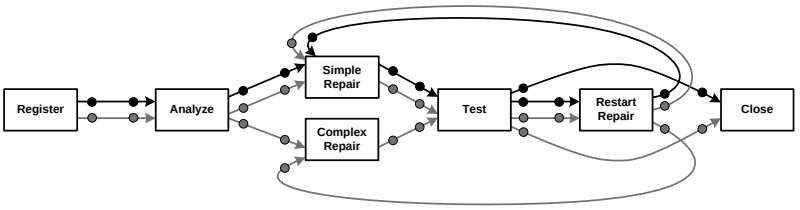


Figure 6.2: Repair process by product type (as workflow constraint).

Depending on the minimum support threshold, the generated amount of itemsets may be excessive. This happens because an itemset property that states that *all nonempty subsets of a frequent itemset must also be frequent.* This is known as Apriori property and implies that the support of all subsets of a specific itemset I is greater or equal than the support of I. For example, a frequent itemset $X = \{x_1, ..., x_{10}\}$ of length 10 can be decomposed in $2^{10} - 2 = 1022$ frequent sub-itemsets (i.e., 10 frequent sub-itemsets of length 1, 45 frequent sub-itemsets of length 2, ..., and 10 frequent sub-itemsets of length 9). A simple solution for this is achieved by introducing some extra itemset properties. Basically, the idea of these extra properties aims at the selection of representative itemsets.

Definition 6.3 (Itemset Properties) *An itemset I can be characterized by the following properties:*

- ***Apriori:*** *The support of the itemset I is greater or equal than the support of all of its super-itemsets.*[1]
- ***Parent Itemset:*** *The itemset I is a* parent *of the itemset J if $J \subset I$ and $|I \setminus J| = 1$.*
- ***Closed Itemset:*** *The itemset I is* closed *if I does not have any parent itemset J such that both I and J have the same support.*
- ***Closed Frequent Itemset:*** *The itemset I is a* closed frequent itemset *if I is both closed and frequent.*
- ***Maximal Frequent Itemset:*** *The itemset I is a* maximal frequent itemset *if I is frequent and there is no other frequent itemset J such that $I \subset J$.*

Remark that the itemset properties also apply to subsequences.

The Apriori property can be used to reduce the search space in the generation of frequent itemsets (or subsequences). This means that *if an itemset I is not frequent then every super-itemset of I is also not frequent* (i.e., the Apriori property is anti-monotone). Therefore, the process of finding frequent itemsets starts with the generation of single-element itemsets. Then, after discarding all non-frequent itemsets, two-element itemsets are generated by combining pairs of single-element frequent itemsets. The process of generating frequent itemsets one element bigger repeats until no further itemsets can be generated. Note that every itemset is discarded after its generation if it is considered non-frequent. This process follows the *candidate generation* approach for finding frequent itemsets and is based on the Apriori algorithm [7]. The pseudo-code for this algorithm and its related functions is provided in Algorithm 8.

[1] J is a super-itemset of I if $I \subset J$.

Algorithm 8: Apriori

Input : an hashmap map $= [(i_1 \mapsto T_1), ..., (i_n \mapsto T_n)]$ in which each entry maps the item i_x to the set of identifiers T_x, and the minimum support threshold.

Output: The frequent itemsets in map.

Step 1
$L_1 \Leftarrow \emptyset$;
foreach $(item \mapsto TIDs)$ *in* map **do**
  support $\Leftarrow$ count$(TIDs)$;
  if support $\geq$ threshold **then** add $\{item\}$ to L_1;
end
return $L_1 \cup$ execute Step 2;

Step n
$L_n \Leftarrow \emptyset$;
candidates $\Leftarrow$ generateCandidates(L_{n-1});
foreach *itemset* $i \in$ candidates **do**
  support $\Leftarrow$ count$(\bigcap_{x \in i}$ map$.get(x))$;
  if support $\geq$ threshold **then** add $\{i\}$ to L_n;
end
if $L_n \neq \emptyset$ **then return** $L_n \cup$ execute Step $n+1$;

function generateCandidates(L)
$newCandidates \Leftarrow \emptyset$;
$knownItemsets \Leftarrow []$;
foreach *itemset* $i \in L$ **do**
  foreach *itemset* j *in* $knownItemsets$ **do**
    $candidate \Leftarrow i \cup j$;
    if *candidate is parent of* i *and* j **then** add *candidate* to $newCandidates$;
  end
  add i to $knownItemsets$;
end
return $newCandidates$;

Association rules

Based on frequent itemsets (or subsequences), association rules are process patterns that state that two – nonempty – groups of event occurrences or dependency relations are associated in terms of appearance in the same process instances. For example, considering again the product repair process in Figure 6.2, the set of event occurrences $\{Test\}$ is associated with the sets $\{Simple\ Repair\}$ and $\{Complex\ Repair\}$. The association should be read as "if a repair is executed then a test is also executed".

Definition 6.4 (Association Rule) *An* association rule $(A \Rightarrow B)$ *expresses that a nonempty itemset A implies the appearance of another nonempty itemset B, with some support (probability) and confidence. Being* $A = \{a_1, ..., a_n\}$ *and* $B = \{b_1, ..., b_m\}$*, an association rule should be read as "*if a_1 *and ... and* a_n then b_1 *and ... and* b_m*".*

Remark that the same logic applies to subsequences.

The process of building association rules relies on frequent itemsets (or subsequences). Hence, before this process starts, the Apriori algorithm needs to be executed in order to find all frequent itemsets. Then, each itemset is decomposed in all possible nonempty subsets. Each of these subsets will form the head of the rule (i.e., the itemset that implies the appearance of another itemset). The remaining part of the rule is formed by the substraction of the original itemset and the head of the rule. Finally, when a rule is built, some measurements are computed in order to quantify the rule interestingness. Eventually, the rule is discarded if it is considered not interesting (or relevant).

There are two basic measures of rule interestingness: *support* and *confidence*. The support of a rule $(A \Rightarrow B)$ consists of the support of the itemset (or subsequence) from which the rule was generated (i.e., the itemset $A \cup B$). The confidence of a rule $(A \Rightarrow B)$ consists of the ratio of the number of process instances where both A and B appear to that where simply A appears.

$$support(A \Rightarrow B) = P(A \cup B) \tag{6.2}$$

$$confidence(A \Rightarrow B) = P(A|B) = \frac{support(A \cup B)}{support(A)} \tag{6.3}$$

Support and confidence are often used to quantify the significance and accuracy of a given rule. However, by simply using these two measures, the amount of generated rules may be excessive. Additionally, some of these rules may not be relevant or add new information. Therefore, we introduce the *lift*, a measure that takes into account the correlation between the itemsets that define the rule (i.e., the correlation between A and B in $(A \Rightarrow B)$). The lift of a rule $(A \Rightarrow B)$ consists of the ratio of the number of process instances where both A and B appear to that if A and B were independent. Hence, if the lift value is 1 then A and B are independent. A lift value greater than 1 indicates that A and B are positively correlated.

$$lift(A \Rightarrow B) = \frac{P(A \cup B)}{P(A) \times P(B)} \tag{6.4}$$

For an overview of measures of interestingness for association rules see [105].

The process of building association rules is summarized as follows:

0. Generate the frequent itemsets.
1. For each frequent itemset i, generate all nonempty subsets of i.
2. For every nonempty subset j of i, build the rule $(A \Rightarrow B)$ with $A = j$ and $B = i - j$.
3. Return all the rules $(A \Rightarrow B)$ that satisfy all of the following conditions:
   - $support(A \Rightarrow B) \geq \tau_{sup}$, with τ_{sup} being a given minimum support threshold
   - $confidence(A \Rightarrow B) \geq \tau_{conf}$, with τ_{conf} being a given minimum confidence threshold
   - $lift(A \Rightarrow B) \geq 1.0$

Gradient rules

From OLAP Mining, gradient rules are process patterns that identify changes of complex measures in multidimensional process models. This kind of process pattern can be seen as a generalization of association rules once that it handles other measurements besides the frequency. Basically, gradient rules show the differences between comparable multidimensional objects with respect to a specific measure of interest. Multidimensional objects can be process instances, event occurrences, dependency relations, or event occurrence bindings.

Definition 6.5 (Multidimensional Relationships) *Let A and B be multidimensional objects of the same type (i.e., process instances, event occurrences, or dependency relations), characterized by the dimension values $D_A = \{a_1, ..., a_n\}$ and $D_B = \{b_1, ..., b_m\}$, with $D_A, D_B \in D$. The objects are*

- ***Parent/Child*** *if $D_X \subset D_Y$ and $|D_Y| - |D_X| = 1$, with $X, Y \in \{A, B\}$ and Y being parent of X;*
- ***Siblings*** *if $|D_A| = |D_B|$ and $|D_A \cup D_B| - |D_A| = 1$.*

A and B are comparable if they are siblings or one is parent of the other.

Definition 6.6 (Gradient Rule) *Let A and B be multidimensional objects of the same type that can be compared (cf. Definition 6.5). A* gradient rule *$(A \succ_f B) = \alpha$ states that $f(A)$ is α times higher than $f(B)$, and is defined as follows:*

$$(A \succ_f B) = \frac{f(A)}{f(B)}, \tag{6.5}$$

where f : $Object \rightarrow \mathbb{R}$ is a summarization function.

In order to demonstrate how gradient rules are computed, we use an illustrative example where some gradient rules on event occurrences are computed. Table 6.2 describes a list of six events occurrences measured by average execution time. Note that these event occurrences are characterized by three dimensions: *Activity*, *Product*, and *Resource*.

#	*Event Occurrence*	*Execution Time*
E_1	{*Activity*:*Analyze*, *Product*:*X*, *Resource*:*Peter*}	5.75 min
E_2	{*Activity*:*Analyze*, *Product*:*Y*, *Resource*:*John*}	6.5 min
E_3	{*Activity*:*Repair*, *Product*:*Y*, *Resource*:*John*}	9.3 min
E_4	{*Activity*:*Repair*, *Product*:*Y*, *Resource*:*Peter*}	9.45 min
E_5	{*Activity*:*Repair*, *Product*:*X*, *Resource*:*Peter*}	11.1 min
E_6	{*Activity*:*Repair*, *Product*:*X*, *Resource*:*John*}	12.5 min

Table 6.2: List of six event occurrences measured by their average execution times.

Let us take the event occurrence E_6 as reference for computing all possible gradient rules from Table 6.2 (i.e., all possible $(E_6 \succ_f i_x)$ or $(i_x \succ_f E_6)$ values where function f measures the average execution time of an event occurrence). According to Definition 6.5, E_1, E_2, and E_4 cannot be compared to E_6 because

- $|D_{E_6} \cup D_{E_1}| - |D_{E_6}| = 5 - 3 \neq 1$,
- $|D_{E_6} \cup D_{E_2}| - |D_{E_6}| = 5 - 3 \neq 1$, and

- $|D_{E_6} \cup D_{E_4}| - |D_{E_6}| = 5 - 3 \neq 1$.

All the other event occurrences can be compared to E_6. Hence, one possible gradient rule is $(E_6 \succ_f E_3) = 12.5/9.3 = 1.34$, which means that the average throughput time of E_6 is 1.34 times higher than the average execution time of E_3. Remark that E_6 may also form a gradient rule with a event occurrence in a higher level of abstraction (e.g., the event occurrences $\{Activity{:}Repair, Product{:}X\}$, $\{Activity{:}Repair, Resource{:}John\}$, and $\{Product{:}X, Resource{:}John\}$).

One of the issues of computing gradient rules is the curse of dimensionality (see Section 5.4). Hence, in order to reduce the search space, a gradient-based cubing strategy to evaluate interesting gradient regions in multidimensional spaces is considered. Introduced in [9], this strategy consists of partitioning the search space into two distinct regions. On the one hand, the *positive* region holds every case with measure value greater or equal than the average of measure values of all cases in the dataset. On the other hand, the *negative* region holds all the cases that do not fit in the positive region. Using some statistics for the different dimensions, gradient rules can then be incrementally generated by selecting pairs of cases from opposite regions.

The process of computing the top-k gradient rules from a dataset of multidimensional objects can be described as follows.

1. Compute the measure value of every object in the dataset using a given function $f : Object \to \mathbb{R}$.
2. Create two regions of objects. The first, the positive region, consists of all objects with measure value greater or equal than the average of measure values of the entire dataset. The second, the negative region, holds the objects that do not fit in the other region.
3. For each region
   (a) Determine all the distinct dimension values characterizing the objects in the region.
   (b) For each distinct dimension value x, compute $bin_x = [min; max]$: the minimum and maximum measure values of the objects characterized by the dimension value.
4. Compute all possible pairs of dimension values (x, y), ordering them by the interval difference of the intersection of bin_x and bin_y. The intersection of $bin_x = [x_1; x_2]$ and $bin_y = [y_1; y_2]$ is defined by $bin_{x \cap y} = bin_x \cap bin_y = [min(x_1, y_1); max(x_2, y_2)]$.
5. For each pair of dimension values (x, y) (considering the ordering determined in the previous line)
   (a) Stop if max/min is not greater than the magnitude of the current k^{th} gradient rule, where $[min, max] = bin_{x \cap y}$.
   (b) Compute all possible gradients rules $(A \succ_f B)$, where B and A are any objects characterized by x and y.
   (c) Update the list of top-k gradient rules with the computed gradients.
6. Return the computed gradient rules.

6.2.4 Classification

Like the rule learning, the classification does not necessarily rely on summarization. Working on a multidimensional process model, the classification function finds relationships that characterizes and discriminates specific aspects described in the model. Depending on the classification approach, these relationships can be expressed as decision trees, business rules, neural networks, etc. In this characterization of classification functions, we will focus on one of the most common classifier techniques: the *decision tree induction*.

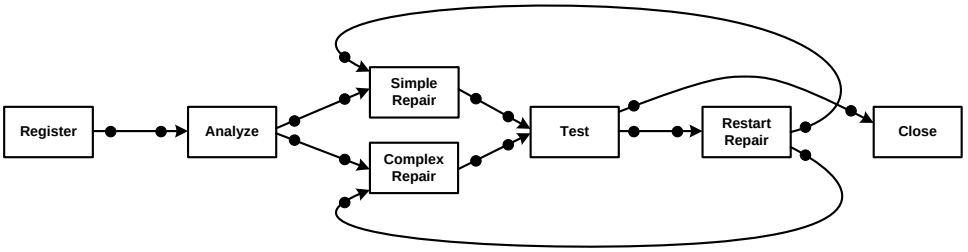
Decision trees

Decision tree induction is one of the most common classifier techniques and basically consists of learning of decision trees from a training dataset (i.e., a set of cases that can be used to train a classifier). A decision tree is a flowchart-like structure consisting of three types of objects: *internal nodes* (or simply *nodes*), *branches*, and *leaf nodes* (or simply *leaves*). Each node represents a test (or decision) on a specific characteristic, each branch identifies the outcome of the test, and each leaf defines the target value for the business rule. The root node is the topmost node (or leaf) of the tree. Figure 6.3d shows an example of a decision tree where nodes (tests) are represented by rounded rectangles, while circles represent the leaves (target values).

The business rules represented by a decision tree are process patterns that characterize specific process behavior or decision. The idea is that specific process behavior (or decision) in a multidimensional process model may be explained by information external to the model (i.e., instance- or event-based data not taken into account in the process model but available in the data). For example, considering the product repair process in Figure 6.3a, the choice of type of repair to be performed after the event *Analyze* can be explained by the rule: "*if* the product type is X or the estimated repair time is 30 minutes *then* execute *Simple Repair else* execute *Complex Repair*". Note that information about the product type and the estimated repair time is available in the process data, but it is not considered in the process model.

Let us use the example from Figure 6.3 to introduce the process of learning a decision tree. As mentioned before, this process relies on training dataset to generate the business rules that form the decision tree. This dataset must contain information about the case's characteristics and the case's target value (or class). In this example, a case consists of the process behavior after the event *Analyze*, which means that we are interested in knowing in what the choice of type of repair is based upon. By using the binding information of event *Analyze* (Figure 6.3b), it is possible to determine which process instances support each of the choices. Hence, the different event occurrence bindings (m_1 and m_2) can be used as classes. The case's characteristics can be retrieved from the index by using the process instance identifiers of the bindings. Figure 6.3c shows the training dataset where there are two characterization attributes and the class. Figure 6.3d depicts the result of applying this process on the training dataset (i.e., the decision tree).

The process of learning a decision tree is based on a top-down approach in which the training dataset is recursively partitioned. In each step of this process, either a node or a leaf of the tree is generated. On the one hand, a leaf is generated if (i) the partition only contains one class, (ii) there are no characterization attributes to be exploited, or (iii) the size of the partition is smaller than a given minimum support



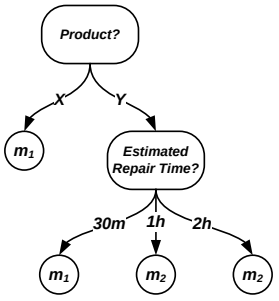
(a) Repair Process.

Binding	*Inputs*		*Outputs*		*Frequency*
			Simple Repair	Complex Repair	
	Register				
m_1	•	$\mapsto$	•		550
m_2	•	$\mapsto$		•	350

(b) Event Occurrence Bindings of *Analyze*.

Product	*Estimated Repair Time*	Class: *Binding*
X	30m	m_1
X	1h	m_1
Y	30m	m_1
Y	1h	m_2
Y	2h	m_2

(c) Training data.



(d) Decision Tree.

Figure 6.3: Induction of a decision tree on event occurrence bindings.

threshold. On the other hand, if none of the previous conditions is satisfied, a node is generated to describe an attribute test condition. The selection of this attribute is based on the concept of entropy used in information theory [101].[2] Two entropy-based measures are considered:

- **Information gain** quantifies the amount of information gained by partitioning a dataset according to the values of a specific attribute. The attribute with highest information gain is the one that needs less information to classify the cases in the dataset, i.e., the one that provides the best partitioning of the dataset where the randomness (or "impurity") is minimized. The information gain of the attribute A ($Gain(A)$) is given by the difference of the expected information needed to classify a case in the dataset D ($Info(D)$) and that on A ($Info_A(D)$). An illustrative example of the information gain computation is provided in Subsection 6.3.4.

[2]Remark that entropy-based attribute selection cannot be performed on continuous-valued attributes. Discretization techniques may be used to overcome this issue. Further details about these can be found in [68].

$$Info(D) = -\sum_{i=1}^{n} \frac{|C_i|}{|D|} \times \log_2(\frac{|C_i|}{|D|}) \tag{6.6}$$

$$Info_A(D) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} \times Info(D_i) \tag{6.7}$$

$$Gain(A) = Info(D) - Info_A(D) \tag{6.8}$$

- **Gain ratio** normalizes the information gain in order to overcome its bias toward attributes with many values. This normalization process relies on an analogous value to $Info(D)$, the *split information*. The gain ratio of the attribute A ($GainRatio(A)$) is given by the ratio of the information gain of the attribute A ($Gain(A)$) to the potential information generated by partitioning the dataset D on A ($SplitInfo_A(D)$). The attribute with highest gain ratio is the one that provides the best partitioning of the dataset. An illustrative example of the gain ratio computation is provided in Subsection 6.3.4.

$$SplitInfo_A(D) = -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \times \log_2(\frac{|D_i|}{|D|}) \tag{6.9}$$

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \tag{6.10}$$

A new partition of the dataset is generated on the selected attribute. This means that the data is partitioned according to the different attribute values. A subtree is generated by recursively applying this learning process on each generated partition and considering the remaining characterization attributes (i.e., the ones not selected yet). The induction of a decision tree using information gain is based on the ID3 algorithm [89]. As an improved version of the ID3, the C4.5 algorithm [90] relies on the gain ratio to generate a decision tree. The pseudo-code for the C4.5 algorithm and its related functions is provided in Algorithm 9.

6.2.5 Clustering

Like other data mining functions, the clustering does not necessarily rely on summarization. Working on a multidimensional process model, the clustering function groups objects (elements) of the process model according to their similarity. A group of objects is designated as *cluster* and comprises objects with common characteristics. Eventually, summarization functions are applied on clusters for characterization. For instance, the average throughput time of a specific cluster of process instances.

There are two main clustering methods:

Partitioning methods distribute the objects by a user-specified number of clusters (K). Clusters are typically represented by a centroid that provides the summary description of all objects in the cluster.

Hierarchical methods generate a hierarchy of clusters in which the objects are distributed from a single cluster comprising all objects to several one-object clusters. Two different approaches can be considered:

Algorithm 9: Decision Tree Generation

Input : The training dataset, the set of characterization attributes, and the minimum support threshold.

Output: A decision tree.

GenerateDecisionTree(dataset,attributes)
  tree $\Leftarrow$ new node;
  if *C is the only class in* dataset **then**
    label tree with the class C;
    return tree as a leaf node;
  end
  if attributes $= \emptyset \vee$ *size of* dataset $<$ threshold **then**
    label tree with the most predominant class in dataset;
    return tree as a leaf node;
  end
  $A \Leftarrow$ `attributeSelection(dataset,attributes)`;
  label tree with A;
  remove A from attributes;
  foreach *value a of attribute A* **do**
    subset $\Leftarrow$ the selection of entries in dataset satisfying the condition $A = a$;
    subtree $\Leftarrow$ the outcome of **GenerateDecisionTree**(*subset*,attributes);
    attach *subtree* to tree;
  end
  return tree;

function attributeSelection(dataset,attributes)
  $bestAttribute \Leftarrow$ null;
  $GainRatio_{bestAttribute} \Leftarrow 0$;
  $Info \Leftarrow -\sum_{i=1}^{n} \frac{|C_i|}{|\text{dataset}|} \times \log_2(\frac{|C_i|}{|\text{dataset}|})$;
  foreach *attribute* $A \in$ attributes **do**
    $Info_A \Leftarrow \sum_{i=1}^{n} \frac{|D_i|}{|\text{dataset}|} \times Info(D_i)$;
    $Gain_A \Leftarrow Info - Info_A$;
    $SplitInfo_A \Leftarrow -\sum_{i=1}^{n} \frac{|D_i|}{|\text{dataset}|} \times \log_2(\frac{|D_i|}{|\text{dataset}|})$;
    $GainRatio_A \Leftarrow \frac{Gain_A}{SplitInfo_A}$;
    if $GainRatio_A > GainRatio_{bestAttribute}$ **then**
      $GainRatio_{bestAttribute} \Leftarrow GainRatio_A$;
      $bestAttribute \Leftarrow A$;
    end
  end
  return *bestAttribute*;

- The *agglomerative approach* starts by considering every object as a cluster and then successively merges the two most similar clusters until only one cluster remains.
- The *divisive approach* starts by considering all objects as a single cluster and

then successively splits a cluster until all clusters consist of a single object.

Note that an object is a cluster that contains one of more process instances, event occurrences, dependency relations, or event occurrence bindings.

Either partitioning or hierarchical methods can be considered to group elements of multidimensional process models. Since the hierarchical methods provide more comprehensive results, we will focus this characterization of clustering functions on this type of clustering method. Further details about agglomerative hierarchical clustering are provided next.

Agglomerative Hierarchical Clustering

Clusters are process patterns that group similar process instances, event occurrences, dependency relations, or event occurrence bindings. For example, the process instances from which a multidimensional process model is built can be grouped into clusters. By merging the most similar clusters one by one, it is possible to create an hierarchy that explains the decomposition of the model into its subparts. More details of this example are provided in Subsection 6.3.5.

The process of clustering multidimensional process model's objects is summarized as follows.

1. Assume that every object is a different cluster.
2. Compute the similarity matrix, i.e., the similarity value for every pair of clusters.
3. **repeat**
   (a) Merge the two most similar clusters.
   (b) Update the similarity matrix with the similarity of the new cluster and the other clusters.
4. **until** only one cluster remains

Computing and maintaining the similarity matrix are the key operations of the clustering function. Relying on the similarity measurements presented in Chapter 4, these operations quantify how similar a specific cluster is to all of the other clusters.

6.3 Extraction of Process Patterns

Considering the data mining functionalities characterized in the previous section, some possible approaches for process pattern extraction are introduced next.

6.3.1 Frequent Event Sets

Frequent event sets are process patterns that identify the event occurrences that are frequently executed together in a process instance. These patterns are particularly useful to identify dependencies among process events, which may not be explicitly shown in the process model. A potential application of these patterns is the identification of independent sub-processes. For instance, taking the process model in Figure 6.4 as reference, it is possible to split the process into two distinct sub-processes. Another application of these patterns is the identification of long-distance dependencies. Further information about these dependencies is provided later in this subsection.

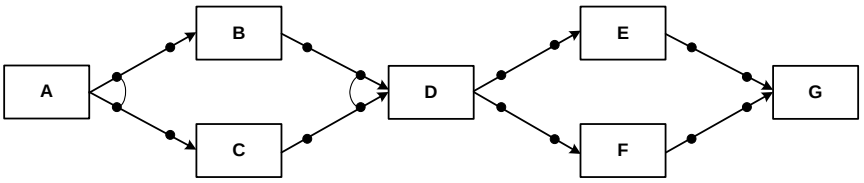


Figure 6.4: A C-net describing a process with parallel behavior.

#	*Process Instance*	*Frequency*
1	$\langle ABCDEG \rangle$	350
2	$\langle ABCDFG \rangle$	300
3	$\langle ACBDEG \rangle$	225
4	$\langle ACBDFG \rangle$	275

Table 6.3: List of all possible process instances that fit in the process of the Figure 6.4.

The frequent event sets of a multidimensional process model can be computed by simply applying the Apriori algorithm presented in Subsection 6.2.3. Table 6.4 presents an explanatory example of the computation of frequent event sets from Table 6.3. Let us assume that there is an index on Table 6.3, and it is possible to identify the exact locations where each event occurrence appears in the process data (i.e., the event occurrences' identifiers). Remark that, for simplicity reasons, events A, B, and C will not be taken into consideration. Considering a minimum support threshold of 50%, the computation process is executed as follows:

1. The first step consists of the identification of distinct event occurrences. Every event occurrence is considered as a single-element event set and, by using the occurrence's identifiers, its support is computed. The outcome of this step is summarized in Table 6.4a. Remark that all event sets satisfy the minimum support threshold.
2. In the second step, the one-element event sets of Table 6.4a are combined into two-element sets. As in the previous step, the supports are computed for every event set. This can be achieved by calculating the intersection of sets of identifiers from the different event occurrences in the event set. Table 6.4b shows all possible two-element event sets and their supports. Remark that the event set $\{E, G\}$ does not satisfy the minimum support threshold. Therefore, this event set is considered as non-frequent and excluded from computation.
3. In the third step, the two-element event sets of Table 6.4b are combined into three-element sets (Table 6.4c). This means that there are cases that cannot be combined because the resulting event set would have more than three elements (e.g., $|\{D, E\} \cup \{F, G\}| = 4$). These cases should not taken into account once that they will be eventually considered in a further step. As in the previous step, the support calculation is performed by intersecting sets of identifiers. Remark that the event sets $\{D, E, F\}$ and $\{E, F, G\}$ will also be discarded because they not satisfy the minimum support threshold.

4. The fourth step is analogous to the third one. Four-element event sets are generated from the ones in Table 6.4c. The only possible four-element event set is described in Table 6.4d. Remark that this is a non-frequent event set. The computation is stopped in this step once that there are no further event sets to be combined.
5. All the frequent event sets found in the four steps of the computation process are described in Table 6.4e. However, some of these event sets are redundant once that do not bring any new knowledge. For instance, the event set $\{D, G\}$ provides the same and further information than both $\{D\}$ and $\{G\}$. Therefore, only maximal frequent event sets (see Definition 6.3) should be considered in the final results. Eventually, in order to provide results with higher supports, closed frequent event sets can also be taken into account. Table 6.4f shows the final frequent event sets. $\{D, E, G\}$ and $\{D, F, G\}$ are maximal frequent event sets, while $\{D, G\}$ is a closed frequent event set.

Event Set	*Support*
$\{D\}$	100%
$\{E\}$	50%
$\{F\}$	50%
$\{G\}$	100%

(a) Step 1.

Event Set	*Support*
$\{D, G\}$	100%
$\{D, E\}$	50%
$\{D, F\}$	50%
$\{E, F\}$	0%
$\{E, G\}$	50%
$\{F, G\}$	50%

(b) Step 2.

Event Set	*Support*
$\{D, E, G\}$	50%
$\{D, F, G\}$	50%
$\{D, E, F\}$	0%
$\{E, F, G\}$	0%

(c) Step 3.

Event Set	*Support*
$\{D, E, F, G\}$	0%

(d) Step 4.

Frequent Event Sets	*Support*
$\{D, G\}$	100%
$\{D\}\{G\}$	100%
$\{D, E, G\}\{D, F, G\}$	50%
$\{D, E\}\{D, F\}\{E, G\}\{F, G\}$	50%
$\{E\}\{F\}$	50%

(e) All frequent event sets.

Frequent Event Sets	*Support*
$\{D, G\}$	100%
$\{D, E, G\}\{D, F, G\}$	50%

(f) Final frequent event sets.

Table 6.4: Finding frequent event sets.

Long-Distance Dependencies

Introduced in Subsection 3.3.1 (p. 71), long-distance dependencies indicate cases where an activity depends indirectly on another activity to be executed. For traditional process models (C-nets), a solution for this issue is given in Algorithm 3. Nonetheless, an alternative solution can be achieved by using frequent patterns such as frequent activity sets (i.e., the activities that are frequently executed together in a process instance).

For multidimensional process models (e.g., mC-nets), a similar process pattern-based solution can be achieved by using frequent event sets.

By transforming frequent event sets into association rules, it is possible to identify dependencies among process events. Instead of indicating that an event depends indirectly on another event to be executed, these association rules indicate that an event depends directly or indirectly on a non-empty set of events to be executed. Long-distance dependencies can be found by filtering out the association rules describing direct dependencies as well as other uninteresting dependencies (e.g., the dependencies between start and end events).

6.3.2 Event Gradients

Event gradients are process patterns that identify changes of measures in the event occurrences of a multidimensional process model. These patterns are particularly useful to compare automatically the performance of event occurrences. For instance, considering the event occurrences described in Table 6.5, it is possible to evaluate the performance of activities and resources in terms of execution time.

#	*Event Occurrence*	*Execution Time*
A	$\{Activity{:}Register, Resource{:}Mary\}$	3.15 min
B	$\{Activity{:}Analyze, Resource{:}Peter\}$	6.5 min
C	$\{Activity{:}Analyze, Resource{:}John\}$	5.75 min
D	$\{Activity{:}Repair, Resource{:}John\}$	8.3 min
E	$\{Activity{:}Repair, Resource{:}Peter\}$	8.65 min
F	$\{Activity{:}Test, Resource{:}Peter\}$	3.1 min
G	$\{Activity{:}Test, Resource{:}John\}$	2.65 min
H	$\{Activity{:}Archive, Resource{:}Mary\}$	3.9 min

Table 6.5: List of eight event occurrences measured by their average execution times.

Computing the top-k event gradients follows the strategy presented in Subsection 6.2.3. Taking the example of the Table 6.5 as reference, the computation of the top-3 gradient rules is described as follows.

1. In the first step, the measure values of every event occurrence are computed using a specific summarization function f. In this example, the measure values are given in Table 6.5.
2. The second step consists of creating the positive and negative regions. Considering that the average of measure values of the entire dataset is 5.25, the regions are defined as:

   R^+ : The event occurrences in the positive region are: B, C, D, and E.

   R^- : The event occurrences in the negative region are: A, F, G, and H.
3. In the third step, the *bins* for every dimension value characterizing the event occurrences in the regions are computed.

   R^+ : The dimension values in the positive region are: $Analyze$, $Repair$, $Peter$, and $John$. The corresponding bins are:
   - $bin^+_{Analyze} = [5.75; 6.5]$

- $bin^{+}_{Repair} = [8.3; 8.65]$
- $bin^{+}_{Peter} = [6.5; 8.65]$
- $bin^{+}_{John} = [5.75; 8.3]$

R^{-} : The dimension values in the negative region are: *Register*, *Test*, *Archive*, *Mary*, *Peter*, and *John*. The corresponding bins are:

- $bin^{-}_{Register} = [3.15; 3.15]$
- $bin^{-}_{Test} = [2.65; 3.1]$
- $bin^{-}_{Archive} = [3.9; 3.9]$
- $bin^{-}_{Mary} = [3.15; 3.9]$
- $bin^{-}_{Peter} = [3.1; 3.1]$
- $bin^{-}_{John} = [2.65; 2.65]$

4. The fourth step consists of computing all combinations of bins, ordering them by the interval difference of their intersection. The top-5 combinations of bins are:
   - $bin^{-}_{Test} \cap bin^{+}_{Repair} = [2.65; 8.65]$
   - $bin^{-}_{Test} \cap bin^{+}_{Peter} = [2.65; 8.65]$
   - $bin^{-}_{John} \cap bin^{+}_{Repair} = [2.65; 8.65]$
   - $bin^{-}_{John} \cap bin^{+}_{Peter} = [2.65; 8.65]$
   - $bin^{-}_{Test} \cap bin^{+}_{John} = [2.65; 8.3]$
5. In the fifth step, the combined bins are used to search gradient rules. Let us use the top combination $bin^{-}_{Test} \cap bin^{+}_{Repair}$ to explain this process. The maximum gradient that can be found in this combination is $8.65/2.65 = 3.26$. bin^{-}_{Test} represents the event occurrences F and G, while bin^{+}_{Repair} represents the event occurrences D and E. Therefore, four candidate rules can be directly formed: $(D \succ_f F)$, $(D \succ_f G)$, $(E \succ_f F)$, and $(E \succ_f G)$. From these, $(D \succ_f F)$ and $(E \succ_f G)$ are discarded because they are not comparable (cf. Definition 6.5). The magnitude of the remaining gradients are then calculated as $(D \succ_f G) = 3.13$ and $(E \succ_f F) = 2.79$. Gradient rules formed by event occurrences in higher levels of abstraction are also searched at this stage. $I = \{Activity{:}Test\}$ and $J = \{Activity{:}Repair\}$ are the only event occurrences in higher levels of abstraction covered by bin^{-}_{Test} and bin^{+}_{Repair}. If not known yet, the measure values for I and J should be computed: $f(I) = 2.88$ and $f(J) = 8.48$. Again, candidate rules can be formed by considering these new event occurrences. $(J \succ_f I) = 2.94$, $(F \succ_f I) = 1.08$, $(I \succ_f G) = 1.09$, $(J \succ_f D) = 1.02$ and $(E \succ_f J) = 1.02$ are the resulting gradient rules. The top-3 gradient rules at the end of this step is $(D \succ_f G) = 3.13$, $(J \succ_f I) = 2.94$, and $(E \succ_f F) = 2.79$. Remark that the other combinations of bins are searched until the maximum gradient that can be found in the combination is smaller than the k^{th} gradient found so far.
6. In the last step, the list with the top-3 gradient rules is returned. There rules can be read as:
   - $(D \succ_f G) = 3.13$: In average, John needs 3.13 times more time to perform a repair than to execute a test.

- $(J \succ_f I) = 2.94$: In average, a repair takes 2.94 times more time than a test.
- $(E \succ_f F) = 2.79$: In average, Peter needs 2.79 times more time to perform a repair than to execute a test.

6.3.3 Frequent Binding Sets

Frequent binding sets are process patterns that identify the bindings that are frequently executed together in a process instance. Like frequent event sets, these patterns also identify dependencies among process events, but at a different level. An event occurrence binding describes a multidimensional relationship of input and output bindings of a specific event occurrence. Hence, frequent binding sets provide insight into frequent behaviors in the business process.

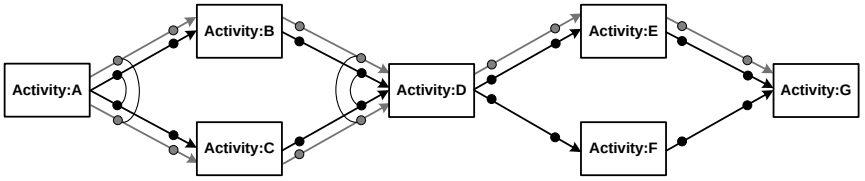


Figure 6.5: A mC-net of the process of the Figure 6.4.

Like frequent event sets (Subsection 6.3.1), the frequent binding sets of a multidimensional process model can be computed by simply applying the Apriori algorithm presented in Subsection 6.2.3. Table 6.8 provides an explanatory example of the computation of frequent binding sets from Table 6.7. Let us assume that there is an index on Table 6.7, and it is possible to identify the exact locations where each binding appears in the process data (i.e., the bindings' identifiers). Remark that, for simplicity reasons, the binding of A, B, and C will not be taken into consideration. Considering a minimum support threshold of 33.3%, the computation process is executed as follows:

1. The first step consists of the identification of distinct bindings. Every binding is considered as a single-element binding set and, by using the binding's identifiers, its support is computed. The outcome of this step is summarized in Table 6.8a. Remark that all binding sets satisfy the minimum support threshold.
2. In the second step, the one-element binding sets of Table 6.8a are combined into two-element sets. As in the previous step, the supports are computed for every binding set. This can be achieved by intersecting sets of identifiers. Table 6.8b shows all possible two-element binding sets and their supports. Remark that the binding sets $\{d_2, e_2\}$, $\{d_2, g_2\}$, $\{d_3, f_1\}$, $\{d_3, g_3\}$, $\{e_2, g_2\}$, and $\{f_1, g_3\}$ are considered as frequent and, therefore, taken into account for further computation.
3. In the third step, the two-element binding sets of Table 6.8b are combined into three-element sets (Table 6.8c). Remark that sets with more than three elements are not considered at this step. As in the previous step, the support calculation is performed by intersecting sets of identifiers. $\{d_2, e_2, g_2\}$ and $\{d_3, f_1, g_3\}$ are the only binding sets considered as frequent in this step.

ID	*Inputs*		*Outputs*		*Frequency*
			B	C	
a_1		$\mapsto$	•	•	750
a_2		$\mapsto$	•	•	400

(a) Bindings of A.

ID	*Inputs*		*Outputs*	*Frequency*
	A		D	
b_1	•	$\mapsto$	•	750
b_2	•	$\mapsto$	•	400

(b) Bindings of B.

ID	*Inputs*		*Outputs*	*Frequency*
	A		D	
c_1	•	$\mapsto$	•	750
c_2	•	$\mapsto$	•	400

(c) Bindings of C.

ID	*Inputs*			*Outputs*		*Frequency*
	B	C		E	F	
d_1	•	•	$\mapsto$	•		175
d_2	•	•	$\mapsto$	•		400
d_3	•	•	$\mapsto$		•	575

(d) Bindings of D.

ID	*Inputs*		*Outputs*	*Frequency*
	D		G	
e_1	•	$\mapsto$	•	175
e_2	•	$\mapsto$	•	400

(e) Bindings of E.

ID	*Inputs*		*Outputs*	*Frequency*
	D		G	
f_1	•	$\mapsto$	•	575

(f) Bindings of F.

ID	*Inputs*			*Outputs*	*Frequency*
	E	F			
g_1	•		$\mapsto$		175
g_2	•		$\mapsto$		400
g_3		•	$\mapsto$		575

(g) Bindings of G.

Table 6.6: The event occurrence bindings of Figure 6.5's mC-Net.

#	*Process Instance*	*Binding Sequence*	*Frequency*
1a	$\langle ABCDEG\rangle^{\bullet}$	$\langle a_1b_1c_1d_1e_1g_1\rangle$	100
1b	$\langle ABCDEG\rangle^{\bullet}$	$\langle a_2b_2c_2d_2e_2g_2\rangle$	250
2a	$\langle ABCDFG\rangle^{\bullet}$	$\langle a_1b_1c_1d_3f_1g_3\rangle$	300
3a	$\langle ACBDEG\rangle^{\bullet}$	$\langle a_1c_1b_1d_1e_1g_1\rangle$	75
3b	$\langle ACBDEG\rangle^{\bullet}$	$\langle a_2c_2b_2d_2e_2g_2\rangle$	150
4a	$\langle ACBDFG\rangle^{\bullet}$	$\langle a_1c_1b_1d_3f_1g_3\rangle$	275

Table 6.7: List of all possible process instances that fit in the process of the Figure 6.5, and corresponding binding sequences.

4. The fourth step ends without any four-element binding set. This happens because the combination of $\{d_2, e_2, g_2\}$ with $\{d_3, f_1, g_3\}$ generates a set with six elements ($\{d_2, d_3, e_2, f_1, g_2, g_3\}$). Therefore, since no valid four-element binding set can be formed, the computation is stopped.
5. All the frequent binding sets found in the computation process are described in Table 6.8d. Table 6.8e shows the maximal frequent binding sets ($\{d_3, f_1, g_3\}$ and $\{d_2, e_2, g_2\}$).

Binding Set	*Support*
$\{d_1\}$	15.2%
$\{d_2\}$	34.8%
$\{d_3\}$	50%
$\{e_1\}$	15.2%
$\{e_2\}$	34.8%
$\{f_1\}$	50%
$\{g_1\}$	15.2%
$\{g_2\}$	34.8%
$\{g_3\}$	50%

(a) Step 1.

Binding Set	*Support*
$\{d_2, d_3\}$	0%
$\{d_2, e_2\}$	34.8%
$\{d_2, f_1\}$	0%
$\{d_2, g_2\}$	34.8%
$\{d_2, g_3\}$	0%
$\{d_3, e_2\}$	0%
$\{d_3, f_1\}$	50%
$\{d_3, g_2\}$	0%
$\{d_3, g_3\}$	50%
$\{e_2, f_1\}$	0%
$\{e_2, g_2\}$	34.8%
$\{e_2, g_3\}$	0%
$\{f_1, g_2\}$	0%
$\{f_1, g_3\}$	50%
$\{g_2, g_3\}$	0%

(b) Step 2.

Binding Set	*Support*
$\{d_2, e_2, g_2\}$	34.8%
$\{d_3, f_1, g_3\}$	50%

(c) Step 3.

Frequent Binding Sets	*Support*
$\{d_3, f_1, g_3\}$	50%
$\{d_3, f_1\}\{d_3, g_3\}\{f_1, g_3\}$	50%
$\{d_3\}\{f_1\}\{g_3\}$	50%
$\{d_2, e_2, g_2\}$	34.8%
$\{d_2, e_2\}\{d_2, g_2\}\{e_2, g_2\}$	34.8%
$\{d_2\}\{e_2\}\{g_2\}$	34.8%

(d) All frequent binding sets.

Frequent Binding Sets	*Support*
$\{d_3, f_1, g_3\}$	50%
$\{d_2, e_2, g_2\}$	34.8%

(e) Final frequent binding sets.

Table 6.8: Finding frequent binding sets.

Sequence Patterns

Sequence patterns are process patterns that describe frequent sequences of bindings in a collection of process instances. This kind of pattern can be seen as a specialization of frequent binding sets once that it identifies sequential relationships on binding sets. Hence, sequence patterns provide insight into frequent behavior sequences in the business process. A potential application of these patterns is the resources management. By identifying frequent sequences of activities, teams can be trained to perform more effectively those activities.

Analogously to computation of frequent bindings, finding sequence patterns relies on the Apriori algorithm to identify frequent binding sequences in a multidimensional process model. The example provided in Table 6.8 can be used to explain the computation process of sequence patterns. Let us assume again that there is an index on Table 6.7, and it is possible to identify the exact locations where each binding appears in the process data (i.e., the bindings' identifiers). Considering the same minimum support threshold of 33.3%, the computation process is executed as follows:

1. Like for frequent binding sets, the first step consists of the identification of distinct bindings. The only difference is that, instead of single-element sets, length-one sequences are considered.
2. In the second step, the length-one sequences are combined into length-two sequences. For frequent binding sets, this can be achieved by intersecting sets of identifiers. For – frequent – sequence patterns, this should be handled in a different way. Intersecting sets of identifiers can be used to select the process instances in which both elements of the sequence appear. If the number of selected process instances does not satisfy the minimum support threshold the length-two sequence is discarded. If not, the actual frequency of the sequence is computed by comparing the positions of the sequence elements in every process instance selected previously. Therefore, the frequency of sequence patterns consists of the number of times the sequence elements appear in a consecutive positioning in the process instances. Remark that the binding sets $\{d_2, e_2\}$, $\{d_2, g_2\}$, $\{d_3, f_1\}$, $\{d_3, g_3\}$, $\{e_2, g_2\}$, and $\{f_1, g_3\}$ are considered as frequent. From these combinations only the binding sequences $\langle d_2e_2\rangle$, $\langle d_3f_1\rangle$, $\langle e_2g_2\rangle$, and $\langle f_1g_3\rangle$ are frequent and, therefore, taken into account for further computation. The sequences $\langle d_2g_2\rangle$ and $\langle d_3g_3\rangle$ are discarded because they do not appear in a consecutive positioning in any process instance.
3. In the third step, the length-two sequences are combined into length-three sequences. The combination of length-n sequences into length-$(n+1)$ sequences consists of finding two length-n sequences X and Y in which the last $n-1$ elements of X are the same as the first $n-1$ elements of Y. The new length-$(n+1)$ sequence is constructed by appending the last element of Y into X (at the end). For example, $\langle d_2e_2\rangle$ and $\langle e_2g_2\rangle$ can be combined into $\langle d_2e_2g_2\rangle$ because the last element of the first sequence is the same as the first element of the second sequence. The same applies to $\langle d_3f_1\rangle$ and $\langle f_1g_3\rangle$. The supports of these new length-three sequences are computed using the same strategy as in the previous step. Remark that both $\langle d_2e_2g_2\rangle$ and $\langle d_3f_1g_3\rangle$ are considered as frequent sequence patterns.
4. Like for frequent binding sets, the fourth step ends without any length-four sequences. This happens because $\langle d_3f_1g_3\rangle$ and $\langle d_2e_2g_2\rangle$ cannot be combined (see

previous step). Since no valid length-four sequences can be formed, the computation is stopped.

5. All the frequent sequence patterns found in the computation process are described in Table 6.9a. Table 6.9b shows the maximal frequent sequence patterns ($\langle d_3 f_1 g_3 \rangle$ and $\langle d_2 e_2 g_2 \rangle$).

Frequent Process Patterns	*Support*
$\langle d_3 f_1 g_3 \rangle$	50%
$\langle d_3 f_1 \rangle \langle f_1 g_3 \rangle$	50%
$\langle d_3 \rangle \langle f_1 \rangle \langle g_3 \rangle$	50%
$\langle d_2 e_2 g_2 \rangle$	34.8%
$\langle d_2 e_2 \rangle \langle e_2 g_2 \rangle$	34.8%
$\langle d_2 \rangle \langle e_2 \rangle \langle g_2 \rangle$	34.8%

(a) All frequent sequence patterns.

Frequent Process Patterns	*Support*
$\langle d_3 f_1 g_3 \rangle$	50%
$\langle d_2 e_2 g_2 \rangle$	34.8%

(b) Final frequent sequence patterns.

Table 6.9: Finding sequence patterns.

6.3.4 Binding Classification

The main idea of *binding classification* is the characterization of bindings. For every event occurrence in a multidimensional process model, the bindings are classified according attributes not represented in the model but available in the index. An illustrative example of this type of process pattern is given in Figure 6.3. Focusing on the event occurrence *Analyze* in Figure 6.3a, this example intends to build a decision tree to explain the execution of the existing bindings (Figure 6.3b). This can be done by (i) generating a learning dataset through the index, and (ii) applying the decision tree generation algorithm (i.e., Algorithm 9) on that dataset. Hence, using the bindings as predicting classes and the *Product* and *Estimated Repair Time* as characterization attributes (Figure 6.3c), it is possible to generate a decision tree (Figure 6.3d) that characterizes the behavior of the process for that specific part.

A learning dataset refers exclusively process data related to a specific event occurrence. The main reference of these datasets is the binding information, which is used as predicting class. Using the references that identify in which process instances the bindings occur (i.e., the bindings' identifiers), it is possible to retrieve – using the index – information about attributes related to the classes to be predicted. Being considered as characterization attributes in the learning dataset, these attributes can be either event- or instance-based attributes.

Building a decision tree follows the strategy presented in Subsection 6.2.4. Taking the example of the Figure 6.3 as reference, we intend to explain the computation process of building a decision tree for the bindings of the event occurrence *Analyze* (Figure 6.3b). Considering the information of the Figure 6.3c as learning dataset and a minimum support threshold of 50%, the decision tree generation process (Algorithm 9) is executed as follows:

1. The decision tree generation process starts by creating the root node of the tree. $\{Product, Estimated\ Repair\ Time\}$ is the set of characterization attributes and $\{m_1, m_2\}$ is the set of classes in the learning dataset. Hence, none of the first two conditions of Algorithm 9 (lines 2 and 6) is satisfied. Then, the characterization

attribute with highest – information – gain ratio is selected. For the attribute *Product*, this measure is computed as follows:

(a) First, the information gain needed to classify a tuple in the dataset is computed.

$$Info(D) = -\frac{550}{900}\log_2\frac{550}{900} - \frac{350}{900}\log_2\frac{350}{900} = 0.964 \text{ bits}$$

(b) Next, assuming that the dataset is partitioned according to the attribute *Product*, it is computed the expected information needed to classify a tuple in the dataset and the gain in information from such partitioning.

$$\begin{aligned} Info_{Product}(D) &= \frac{400}{900} \times (-\frac{400}{400}\log_2\frac{400}{400} - \frac{0}{400}\log_2\frac{0}{400}) \\ &\quad + \frac{500}{900} \times (-\frac{150}{500}\log_2\frac{150}{500} - \frac{350}{500}\log_2\frac{350}{500}) \\ &= 0.49 \text{ bits} \end{aligned}$$

$$Gain(Product) = Info(D) - Info_{Product}(D) = 0.964 - 0.49 = 0.474 \text{ bits}$$

(c) Then, a normalization operation is applied to information gain in order to minimize the bias towards tests with many outcomes. Finally, the gain ratio can be determined.

$$SplitInfo_{Product}(D) = -\frac{400}{900}\log_2\frac{400}{900} - \frac{500}{900}\log_2\frac{500}{900} = 0.991$$

$$GainRatio(Product) = \frac{Gain(Product)}{SplitInfo_{Product}(D)} = \frac{0.474}{0.991} = 0.478$$

A summary of the calculation process of the gain ration for both attributes *Product* and *Estimated Repair Time* is provided in Table 6.10. Since *Product* is the attribute with highest gain ration, this attribute should be removed from the set of characterization attributes, and the root node should be labeled with the text "*Product*?". Finally, for every attribute value v of attribute *Product* (i.e., X and Y), a subtree based on the remaining characterization attributes and a subset of the learning dataset containing the entries that satisfy the condition $Product = v$ will be generated and attached to the root node.

2. The subtree representing the test $Product = X$ is generated by executing the method **GenerateDecisionTree** on the subset of the learning dataset containing the entries that satisfy the condition $Product = X$ (i.e., the first two rows of the Figure 6.3c). The subtree generation process starts by creating the root node of the subtree. Since m_1 is the only class in this subset, the first condition of the algorithm (line 2) is satisfied. Therefore, the root node is labeled with the text "m_1" and converted into a leaf node. The process is terminated by returning the subtree.

3. The subtree representing the test $Product = Y$ is generated by executing the method **GenerateDecisionTree** on the subset of the learning dataset containing the entries that satisfy the condition $Product = Y$ (i.e., the last three rows of the Figure 6.3c). The subtree generation process starts by creating the root node of the subtree. $\{Estimated\ Repair\ Time\}$ is the set of characterization attributes, $\{m_1, m_2\}$ is the set of classes in the learning dataset, and $55,56\%$ is the – relative – size of the subset. Hence, none of the first two conditions of the algorithm (lines 2 and 6) is satisfied. Then, the attribute *Estimated Repair Time* is selected and removed from the set of characterization attributes. Remark that, since this is the only attribute available, there is no need to compute the gain ratio measure. After labeling the subtree's root node with the text "*Estimated Repair Time*?", further subtrees will be generated for every attribute value v of attribute *Estimated Repair Time* (i.e., 30m, 1h, and 2h), and attached to the root node. The subsets of the learning dataset for these new subtrees should contain the entries that satisfy the conditions $Product = Y$ and $Estimated\ Repair\ Time = v$. Since any of these subsets satisfies the minimum support threshold, all of these new subtrees will be defined by leaf nodes representing the most predominant class of each subset.

Attribute Measure	*Product*	*Estimated Repair Time*
$Info_A(D)$	0.49	0.496
$Gain(A)$	0.474	0.468
$SplitInfo_A(D)$	0.991	0.981
$GainRatio(A)$	**0.478**	0.477

Table 6.10: Summary of attribute measures for *Product* and *Estimated Repair Time*.

6.3.5 Instance Clustering

The main idea of *instance clustering* is the aggregation of similar process instances. Taking a multidimensional process model as reference, process instances (i.e., traces) are grouped according to similarity.

Definition 6.7 (Process Instance Equivalence) *Let a process instance T be a finite sequence of process events $\langle e_1 e_2 ... e_n \rangle$ that can be translated into the sequence of event occurrence bindings $T' = \langle m_1 m_2 ... m_n \rangle$. The equivalence of two process instances T_1 and T_2 is determined by the appearance of bindings in T_1' and T_2', i.e., every binding in T_1' must appear the same number of times in T_2' and vice versa. This can be checked through the function* $\mathrm{id}(T')$ *which transforms a sequence of event occurrence bindings into a multiset of event occurrence bindings. Two process instances T_1 and T_2 are considered equivalent if $id(T_1') = id(T_2')$.*

The following example is used to demonstrate the rationale behind this definition. Let us assume that there are two binding sequences $T_1' = \langle a_1 b_1 c_1 d_1 e_1 g_1 \rangle$ and $T_2' = \langle a_1 c_1 b_1 d_1 e_1 g_1 \rangle$. Due to the different ordering of bindings (i.e., b_1 and c_1 appear in a different order in different sequences), T_1' and T_2' cannot be considered equivalent simply by comparing the sequences. However, since event occurrence bindings define

the behavior of event occurrences (i.e., the relationship of activated inputs and outputs of a specific event occurrence), the ordering in binding sequences become irrelevant. Therefore, checking that every binding in a sequence appears the same number of times in another (i.e., $id(T'_1) = id(T'_2)$) is enough to determine equivalence between binding sequences. So, one can conclude that T_1 and T_2 are equivalent once that $id(T'_1) = id(T'_2) = [a_1, b_1, c_1, d_1, e_1, g_1]$.

A cluster of process instances is a group of one or more process instances. Considering the set representation of process instances (i.e., the multiset of event occurrence bindings obtained from function $id(T')$ in Definition 6.7), the aggregation of process instances can be simply defined by the union of the instances' event occurrence bindings in which their frequencies are taken into account. The result of aggregating two process instances is also a multiset of event occurrence bindings, and can be seen as a cluster containing those two instances. The aggregation of clusters of process instances is analogous to the definition provided above in this paragraph. Remark that a process instance can be seen as a single-element cluster.

Definition 6.8 (Process Instances Aggregation) *Let $A = [a_1^{\alpha_1}, ..., a_n^{\alpha_n}]$ and $B = [b_1^{\beta_1}, ..., b_m^{\beta_m}]$ be process instances or clusters of them. The aggregation of A and B is defined by the sum of A and B (i.e., $A + B$).*

The following example is used to explain how process instances (or clusters of them) can be aggregated. Let us consider the following single-element clusters of process instances from Figure 6.6: $C_A = [a_1, b_1, c_1, d_1, e_1, g_1]$ and $C_B = [a_1, b_1, c_1, d_3, f_1, g_3]$. Applying the definition provided above, the result of aggregating C_A and C_B is:

$$\begin{aligned} C_{AB} = C_A + C_B &= [a_1, b_1, c_1, d_1, e_1, g_1] + [a_1, b_1, c_1, d_3, f_1, g_3] \\ &= [a_1^2, b_1^2, c_1^2, d_1, d_3, e_1, f_1, g_1, g_3] \end{aligned}$$

The similarity between clusters of process instances is calculated according to Definition 4.4. Therefore, applying this definition, the similarity matrix of the instance clusters in Figure 6.6 is presented in Table 6.11. As depicted in the figure, C_A, C_B and C_C are single-element clusters (or process instances), C_{AB} is a two-element cluster representing C_A and C_B, and C_{ABC} is a three-element cluster representing C_A, C_B and C_C.

	C_A	C_B	C_C	C_{AB}	C_{ABC}
C_A	1.0	0.89	0.72	0.31	0.23
C_B	0.89	1.0	0.61	0.31	0.22
C_C	0.72	0.61	1.0	0.22	0.23
C_{AB}	0.31	0.31	0.22	1.0	0.47
C_{ABC}	0.23	0.22	0.23	0.47	1.0

Table 6.11: Similarity matrix of the instance clusters in Figure 6.6.

The instance clusters of a multidimensional process model can be computed by applying an agglomerative hierarchical clustering method (see Subsection 6.2.5). Table 6.12 provides an explanatory example of the clusters computation on Table 6.7's process instances. The computation process is divided by three steps that are described as follows:

1. The first step consists of the identification of distinct process instances. Therefore, the list of six process instances is reduced to three once that there are three pairs of process instances that can be considered equivalent: (1a,3a), (2a,4a) and (1b,3b). Each one of these three process instances is considered as a single-element cluster. The instance clusters generated in this step are described in Table 6.12a. To finalize this step, it is necessary to build a similarity matrix, which means that the similarity value for every pair of clusters must be computed. The similarity between these clusters is calculated according to the approach introduced in Chapter 4. $sim(C_A, C_B) = 0.89$, $sim(C_A, C_C) = 0.72$ and $sim(C_B, C_C) = 0.61$ are the similarity values (Table 6.11).
2. In the second step, the the two most similar clusters are merged. So, clusters C_A and C_B are integrated in a new cluster C_{AB} (Table 6.12b). The similarity matrix needs to be updated with the similarity values of the new cluster and all other remaining clusters. $sim(C_C, C_{AB}) = 0.22$ is the new similarity value (Table 6.11).
3. The third and final step is analogous to the second one. Clusters C_C and C_{AB} are integrated in a new cluster C_{ABC} (Table 6.12c). The computation is stopped in this step once that there are no further clusters to be aggregated.

Remark that some summarization function may be applied on a specific cluster (e.g., to describe the cluster's frequency). This can be achieved by using the process instances' identifiers.

Cluster	*Process Instances*	*Frequency*
C_A	1a/3a	175
C_B	2a/4a	575
C_C	1b/3b	400

(a) Step 1.

Cluster	*Process Instances*	*Frequency*
C_{AB}	1a/3a and 2a/4a	750
C_C	1b/3b	400

(b) Step 2.

Cluster	*Process Instances*	*Frequency*
C_{ABC}	1a/3a, 2a/4a and 1b/3b	1150

(c) Step 3.

Table 6.12: Computing instance clusters.

The final result of computing instance clusters is an hierarchy that explains the decomposition of the mC-net of the Figure 6.5 into its subparts. This cluster hierarchy is depicted in Figure 6.6. The cluster at the top represents the aggregation of all process instances and can be seen as the entire process model. At the bottom, all distinct process instances are identified. Middle layers represent partial aggregations of clusters. The solid lines define which clusters are being aggregated.

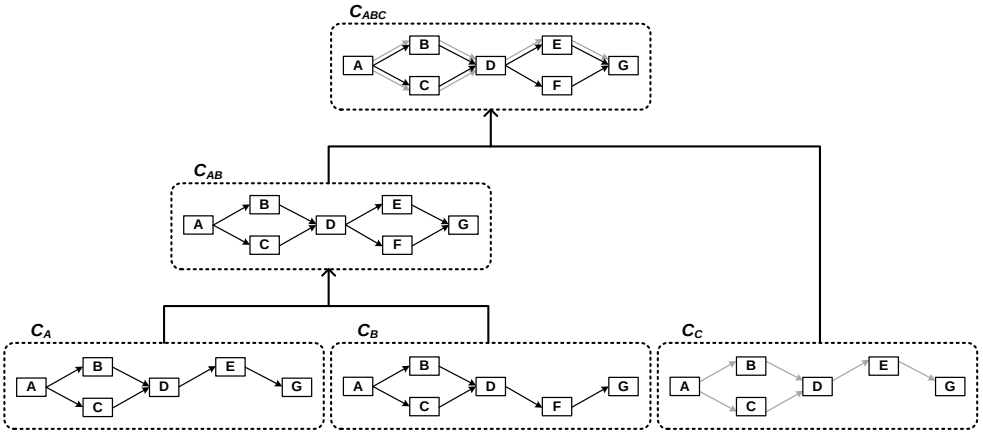


Figure 6.6: Cluster hierarchy of the mC-net of the Figure 6.5.

6.4 Summary

Two components of the multidimensional process discovery approach were discussed in this chapter.

- Characterized in Section 6.1, **process patterns** consist of any kind of pattern extracted from a multidimensional process model. Instance-, event-, and flow-based patterns are the three types of process patterns described in this thesis.
- **Process patterns miners** are introduced in Section 6.3 as a set of possible knowledge discovery techniques for extracting process patterns from multidimensional process models. By applying data mining functionalities on process data, it is possible to get further insight into the process behavior described in a multidimensional process model.

As instances of the process patterns miners component, the instance clustering and the frequent event sets techniques are implemented as objects that can be executed in the ProM framework [123, 125].

Answering research questions q_4 and q_5, the extraction of process patterns from multidimensional process models and the hypothesis-driven analysis (discussed in the previous chapter) are two approaches for knowledge discovery on multidimensional process models. Instance-, event-, and flow-based patterns are possible patterns that can be extracted from a multidimensional process model.

Chapter 7

Evaluation

In the previous chapters, we introduced the Multidimensional Process Explorer framework, an approach for process discovery and analysis taking into account the dimensionality of business processes. In this chapter, we present a study that aims at the evaluation of that approach. By conducting a set of experiments as well as a case study in real-life environments, it is intended to assess

i. the efficiency of the different components of the framework (experiments),

ii. the types of problems that can be solved by applying the proposed approach (experiments and case study), and

iii. the perception the users have after using the framework (experiments and case study).

Remark that the main goal of this chapter is the evaluation of the multidimensional framework. Once that no alternative approach is known, it is not possible to perform a comparative study of the performance of the Multidimensional Process Explorer framework. Hence, a software quality model is considered to evaluate the framework.

In the next sections, we firstly introduce the software quality model used as basis of the evaluation study. Then, we describe an implementation of the Multidimensional Process Explorer framework as a prototype. Next, we explain how the considered aspects of the software quality model are evaluated. Finally, we present and discuss the evaluation results.

7.1 Software Quality Model

In order to evaluate the Multidimensional Process Explorer framework, a model for assessing the quality of software is introduced in this section: the ISO 9126-1 Software Quality Model [1, 79]. Six software quality characteristics are considered in this model.

Functionality determines the capability of the framework to provide functions which satisfy stated and implied needs when it is used under specified conditions. The framework functionality can be evaluated according to four aspects.

- **Suitability**: the capability of providing the appropriate functionality for the required tasks.

- **Accuracy**: the capability of providing the correct results with the needed degree of precision.
- **Interoperability**: the capability of interaction with other systems.
- **Security**: the capability of prevention of unauthorized access to functions or data.

Reliability determines the capability of the framework to maintain its level of performance when used under specified conditions. The framework reliability can be evaluated according to three aspects.

- **Maturity**: the capability of avoiding failures, as a result of faults in the framework.
- **Fault Tolerance**: the capability of maintaining a specified level of performance in case of fault in the framework.
- **Recoverability**: the capability of recovering from failures.

Usability determines the capability of the framework to be understood, learned, and used by a stated or implied group of users. The framework usability can be evaluated according to three aspects.

- **Understandability**: the capability of enabling the users to understand the functionality provided in the framework.
- **Learnability**: the capability of enabling the users to learn the application of the framework.
- **Operability**: the capability of enabling the users to operate and control the framework.

Efficiency determines the capability of the framework to accomplish its functionality in reasonable time considering the available resources. The framework efficiency can be evaluated according to two aspects.

- **Time Behavior**: the response time, processing time and throughput rates of the computational process.
- **Resource Behavior**: the amount of resources required in the computation.

Maintainability determines the capability of the framework to be updated. Updates may include corrections, improvements or adaptations of the framework to changes in the environment and in requirements and functional specifications. The framework maintainability can be evaluated according to four aspects.

- **Analyzability**: the capability of identification of root causes of failures.
- **Changeability**: the capability of incorporation of updates into the framework.
- **Stability**: the capability of avoiding unexpected effects from updates.
- **Testability**: the capability of validation of a specific update.

Portability determines the capability of the framework to work in different environments. The framework portability can be evaluated according to four aspects.

- **Adaptability**: the capability of adaptation to specified environments.
- **Installability**: the capability of installation in a specified environment.
- **Co-Existence**: the capability of co-existence with other independent software in a common environment, sharing common resources.
- **Replaceability**: the capability of replacement of components within a specified environment, without changing the original functionality.

Figure 7.1 presents an overview of ISO 9126-1 Software Quality Model. The characteristics and aspects identified by the gray color are the ones considered in the evaluation of the Multidimensional Process Explorer framework. Basically, this selection takes into account the fact that the current implementation of the framework simply consists of a prototype and not a commercial product. Also, the considered aspects can provide insight into the added value of the proposed framework to the stakeholders.

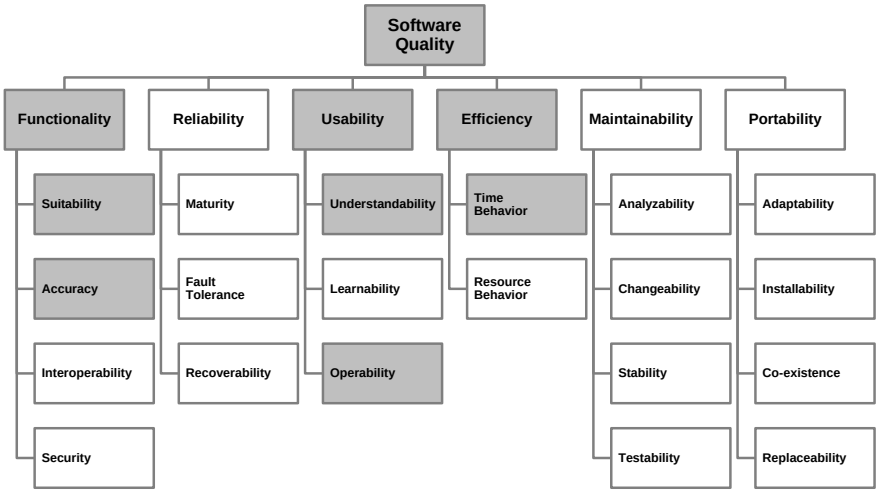


Figure 7.1: Overview of the ISO 9126-1 Software Quality Model.

7.2 Implementation

In this section, we characterize the implementation of the Multidimensional Process Explorer framework. This implementation consists of several components, which are described in the previous chapters. Figure 7.2 presents a component diagram of the implemented components according to the concept architecture presented in Section 2.4 (Figure 2.11). In this diagram, the data provider component is not considered, while the Event Cube and Process Explorer components are represented by the (sub-)components in the green and blue boxes. Representing the traditional process discovery approach, the components in the red box are external to the concept architecture. Remark that, besides the identified components, the augmented Causal-Nets are implemented as an instance of traditional process models, and the multidimensional Causal-Nets are implemented as an instance of multidimensional process models.

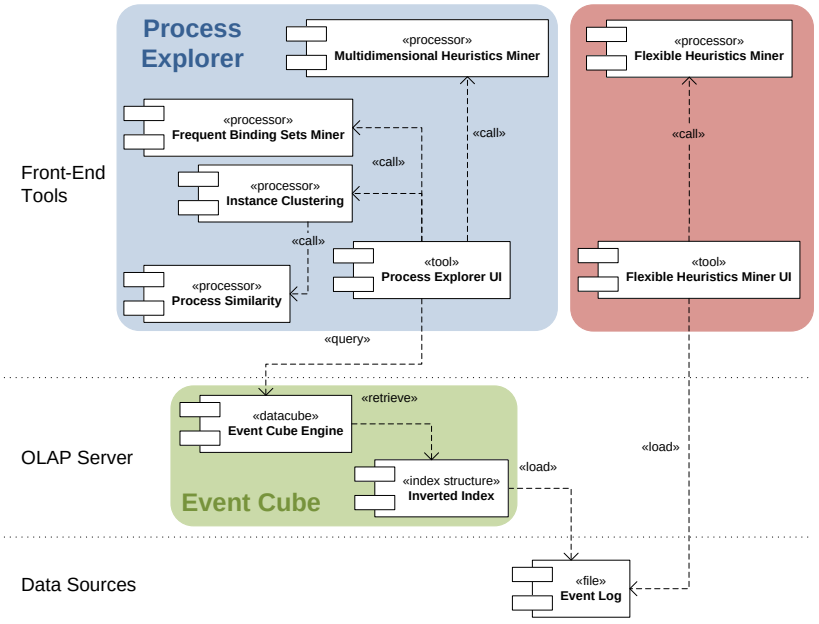


Figure 7.2: Component diagram of the implemented components of the Multidimensional Process Explorer framework.

The implemented components can be characterized as follows.

Augmented Causal-Net (aC-net) is a Java package that implements the process notation defined in Definition 3.6 (see Subsection 3.2.1). The main functions of this component are:

- **Augmented Causal-net constructor**: builds an aC-net from a multiset of input and output bindings.
- **Augmented Causal-net visualizer**: provides a graphical representation of an aC-net.
- **Augmented Causal-net to Petri net converter**: converts an aC-net into a Petri net.

Multidimensional Causal-Net (mC-net) is a Java package that implements the process notation defined in Definition 3.10 (see Subsection 3.2.2). The main functions of this component are:

- **Multidimensional Causal-net constructor**: builds a mC-net from a multiset of event occurrence bindings.
- **Multidimensional Causal-net visualizer**: provides a graphical representation of a mC-net.

Flexible Heuristics Miner (FHM) is a Java package that implements the control-flow mining technique introduced in Subsection 3.3.1. The main function of this component is:

- **FHM processor**: mines an aC-net according to algorithms 1, 2, and 3 (pages 64, 69, and 73).

Multidimensional Heuristics Miner (MHM) is a Java package that implements the control-flow mining technique introduced in Subsection 3.3.2. The main function of this component is:

- **MHM processor**: mines a mC-net according to algorithms 4 and 5 (pages 75 and 80).

Inverted Index is a Java object that implements the information retrieval functionality described in Subsection 5.2.1. The main functions of this inverted index implementation are:

- **Time discretizer**: derives new time categories from the timestamps in the process data (see Subsection 5.2.2).
- **Index processor**: creates an inverted index of all attributes in the process data.
- **Normalizer of event-based attributes**: converts event-based attributes into instance-based ones.
- **Query processor**: retrieves the set of process events or instances that satisfy a given filtering condition (cf. Subsection 5.2.1).

Event Cube is a Java package that implements the data cube described in Section 5.2. The main functions of this component are:

- **Measures**: consist of a set of – predefined – control-flow and performance measures that can be used to generate measuring information of multidimensional cells (see Subsection 5.2.3).
- **Aggregation Functions**: consist of a set of – predefined – aggregation functions that can be used to summarize measuring information of multidimensional cells (see Subsection 5.2.3).
- **Cube processor**: builds a data cube from the process data, as described in Subsection 5.2.4. Remark that this processor applies a multithreading method for computing measuring information.
- **Query processor**: retrieves – from the data cube – the measuring information of a specific cuboid or multidimensional cell.

Process Explorer is a Java package that implements a graphical interface for exploiting an Event Cube (see Subsection 5.3.3). The main functions of this component are:

- **Discovery operators**: define the characterizing information to be provided in the mC-net graphical representation.
- **Filtering operators**: constraint the – characterizing and measuring – information to be provided in the mC-net graphical representation.
- **Measure selector**: identifies the measuring information to be provided in the mC-net graphical representation.
- **Parameter selector**: regenerates on-the-fly the mC-net in analysis according to new parameter settings.

Process Similarity is a Java object in which the similarity measures introduced in Chapter 4 can be computed. The main function of this component is:

- **Similarity processor**: computes the similarity between objects of multi-dimensional process models.

Instance Clustering is a Java package in which clusters of process instances as described in Subsection 6.3.5 can be mined. The main functions of this component are:

- **Miner processor**: computes the instance clusters from a mC-net.
- **Instance cluster visualizer**: provides a graphical representation of an instance cluster.

Frequent Binding Sets Miner is a Java package in which sets of event occurrence bindings that are frequently executed together in a process instance can be mined (cf. Subsection 6.3.3). The main functions of this component are:

- **Miner processor**: computes the frequent binding sets from a mC-net.
- **Binding set visualizer**: provides a graphical representation of a frequent binding set.

Remark that the components of the Multidimensional Process Explorer framework are implemented as ProM 6 plugins, or objects that can be executed in the ProM framework [123, 125].

7.3 Evaluation Method

In this section, we describe the used methodology to assess the five selected characteristics of the software quality model presented in Section 7.1. This methodology aims at the gathering of evaluating information and consists of three different approaches:

i. **Execution data** approach consists of a set of experiments from which the performance aspect of the framework can be evaluated.

ii. **Results** approach consists of a case study where an explorative analysis of a real-life process is conducted. The analysis results are used to get feedback from the business analysts.

iii. **User experience** approach consists of a workshop where different kinds of users can use the framework to discover and analyze a specific repair process. Their feedback is gathered thought a questionnaire.

Remark that these three approaches have direct mapping to the three aspects mentioned in the introduction of this chapter.

Structure

The structure of this evaluation is presented as follows. First, in the next subsections, the different evaluation analyses (performance experiments, case study, and workshop) are introduced. Next, the results of these analyses are presented in sections 7.4, 7.5, and 7.6. Finally, the conclusions of all evaluation analyses as well as a final discussion are provided at the end of this chapter (Section 7.7).

7.3.1 Experiments

An experimental study was conducted to evaluate the efficiency aspect of the Multidimensional Process Explorer framework. Using synthetic process data, the main framework's components are assessed in terms of time behavior in order to identify the system bottlenecks.

Applying the method described in [3], a repair process is defined to support the generation of synthetic process data. Basically, this process is composed by 12 activities in which 12 human resources attempt to repair 3 different kinds of products. Each case is characterized by a problem types (out of 10), which determine how the problem should be handled. The case management is provided by a system, which is – together with the human resources – one of the resources of the process. The generated process data consist of simulation results over the repair process.

Table 7.1 characterizes the 13 datasets (event logs) artificially generated for these experiments. Datasets d_{1-6} vary in terms of size (i.e., process instances and events). Datasets d_{a-g} vary in terms of resources and products in the process. This means that the human resources and the products are multiplied (cloned) by a factor from 2 to 8. Considering the computing time as the main performance indicator, these experiments aim at the analysis of the impact of three main aspects: (i) the size of the dataset, (ii) the Event Cube's dimensionality, and (iii) the dimensions' cardinality. The size aspect (i) will be analyzed using datasets d_{1-6}, which are characterized by different amount of process events. In these datasets, the number of process events per process instance is about the same (around 11 events per instance). The characteristics of the repair process are preserved over the datasets (e.g., the number of activities and resources are the same in all datasets). The aspects related to dimensionality (ii and iii) will be analyzed using datasets d_2 and d_{a-g}, which are characterized by different amount of resources and products. The sizes of these datasets are about the same.

Dataset	*Process Instances*	*Process Events*	*Activities*	*Resources*	*Products*	*Problem Types*
d_1	500	5615	12	12+1	3	10
d_2	1000	11345	12	12+1	3	10
d_3	2000	22895	12	12+1	3	10
d_4	4000	45610	12	12+1	3	10
d_5	8000	91710	12	12+1	3	10
d_6	16000	182110	12	12+1	3	10
d_a	1000	11215	12	24+1	6	10
d_b	1000	11135	12	36+1	9	10
d_c	1000	11025	12	48+1	12	10
d_d	1000	11105	12	60+1	15	10
d_e	1000	11165	12	72+1	18	10
d_f	1000	11155	12	84+1	21	10
d_g	1000	10905	12	96+1	24	10

Table 7.1: The event logs artificially generated.

All the experiments were run on an Intel Core 2 Quad Q9650 3.0GHz machine with 4Gb RAM on Microsoft Windows XP OS. The maximum heap size of the Java Virtual

Machine (JVM) was set to 1Gb. The results of these experiments are presented in Section 7.4.

7.3.2 Case Study

A large global company that designs and manufactures high-end medical systems such as X-ray machines and MRI systems[1] provides technical and customer support for its products. In order to offer appropriate support services under a limited budget, the business analysts have been putting many efforts in optimizing the support services by monitoring and assessing the performance of service cases. Although service cases are described by a sequence of events, the process behavior described in service cases is not considered yet in the performance assessment of support services.

A case study was conducted to assess the performance of technical and customer support services taking into account the process behavior of the service cases. As outcome of this study, a methodology is developed to retrieve process data from a data repository, transform these data into process instances, and integrate these process instances into an event log. Additionally, an explorative analysis of the support services is conducted using the Multidimensional Process Explorer framework. The results of this analysis are used to get feedback about the added value of the multidimensional process discovery approach from the business analysts.

Description and purpose of existing corporate data

The case study aims at the assessment of technical and customer services. Typical examples of service cases are planned system maintenances, remote assistances, or mal-functioning component replacements. All the data about technical and customer services are maintained in a data warehouse, which integrates data from multiple sources. Basically, four main aspects are described in service data: *systems*, *service cases*, *tasks*, and *components*. Information about systems can be seen as instance-based information that describes the system being maintained or repaired in a specific service case. Information about service cases can be seen as process instances. A service case is characterized by one or more tasks, which can be seen as process events. Information about components can be seen as event-based information that describes the components being analyzed, repaired or replaced in a specific task.

Currently, the performance of service cases is assessed using statistical-based tools. So far, this approach has proven to be effective, but it misses the characterization of the business processes. For example, the throughput times of process instances can be easily computed from the existing data. However, the performance assessment of these process instances simply can be done by analyzing the sequences of events that characterize the instances. This means that the business process is assumed exactly as described in the process instances, disregarding eventual low-frequent or parallel behavior. By characterizing the process behavior of service cases, it is possible to extend the current performance assessment with process analysis. In this case study, a potential application of this extension is the comparison of the performance of similar service cases (i.e., cases characterized by similar process behavior) in different markets, for different systems and contract types.

[1] MRI stands for Magnetic Resonance Imaging.

Data preparation and explorative analysis

Firstly, a data analysis was performed to characterize the available data as well as to assess the data quality. Four data views describing the main aspects of the business process (i.e., *systems*, *service cases*, *tasks*, and *components*) can be generated from the data warehousing system. The relationships between these aspects can be described as follows. One service case is performed on one system and is defined by one or more tasks. One task is performed on zero or more components of a specific system. The quality of service data (i.e., integrity and consistency) is ensured by the data warehouse system.

Three steps are necessary to prepare the service data to process discovery and analysis. First, the different aspects of service data should be integrated into a single data view. In this process, only the data to be analyzed should be considered. This means that some attributes can be disregarded. Additionally, filtering conditions can be used to reduce the amount of data (e.g., considering the service data of a specific period). Second, the integrated service data should be exported to a CSV (comma-separated values) file. Third, an event log should be generated from the CSV file. This can be done by using import tools such as ProM Import[2] or Nitro[3].

In this case study, three aspects of the support services are considered: *systems* as instance-based information, *service cases* as process instances, and *tasks* as process events. From the data views describing these aspects, eight attributes are selected for analysis:

- *Case ID* is a unique identifier of a service case. The service data contains about 280000 distinct cases.
- *Task ID* is a categorical attribute that describes the task performed in a specific service case. The service data contains about 1.4 million tasks (44 distinct tasks). This attribute is used as the activity of the process event.
- *Task Beginning* is a timestamp that identifies the beginning of a task. This attribute is used to compute the throughput time of service cases.
- *Task End* is a timestamp that identifies the end of a task. This attribute is used to compute the throughput time of service cases.
- *Market* is a categorical attribute that identifies the market where a specific service was performed. 14 distinct markets can be found in the data. This attribute is considered as instance-based information.
- *System ID* is a categorical attribute that describes the system on which a specific service was performed. 100 distinct systems can be found in the data. This attribute is considered as instance-based information.
- *Contract Type* is a categorical attribute that describes the type of contract associated to a specific system. 8 distinct contract types can be found in the data. This attribute is considered as instance-based information.
- *Cost* is a numerical attribute that describes the total cost of a specific task. This attribute is used as performance measure.

In order to reduce the amount of data, a subset of these service data is generated by applying some filtering conditions. This subset is characterized by 6 systems, 4

[2] `http://www.promtools.org/promimport/`
[3] `http://fluxicon.com/nitro/`

markets, and 5 contract types, characteristics defined with the support of a business analyst. Using this subset, an event log with about 7000 cases and 48000 tasks is finally generated.

The explorative analysis of the service data consists of discovering and analyzing the business processes using the Multidimensional Process Explorer framework. Multidimensional process models are generated to characterize the process behavior of the different business perspectives. Additionally, process patterns related to these process models are also derived. The most representative results are selected to be discussed with a committee formed by five business analysts of the involved company. The goal of this discussion is the assessment of the suitability and accuracy of the presented process models and patterns. The feedback gathered in this discussion is used to determine the added value of the multidimensional process discovery approach. The results of this discussion are presented in Section 7.6.

7.3.3 Workshop

A workshop was organized to demonstrate the functionalities of the Multidimensional Process Explorer framework to an audience composed by both academics and industrials. Two workshop sessions were held at Eindhoven University of Technology in two distinct events: (i) in September 2011, during a meeting in the context of the DataFusion project where the industrial partners were present, and (ii) in April 2012, as part of the process mining course.

The structure of the workshop can be described as follows. First, a presentation was given to the participants in order to introduce the Multidimensional Process Explorer framework as well as related concepts (e.g., multidimensional process models and business process queries). Second, an interactive session where the participants could use the framework to discover and analyze a specific repair process was organized. Finally, a questionnaire was given to the participants in order to gather information about them and their experience with the framework. Further details as well as the results of this experiment are presented in sections 7.5 and 7.6.

Participants Profile

In total, there were 25 participants from which 60% are academics (process mining students) and 40% are industrials (managers or engineers). The following question was posed to the workshop participants in order to characterize them in terms of domains of expertise: *In a scale of 1 (unfamiliar) to 10 (familiar), how familiar are you with the following terms?*

- *Business Intelligence (BI)*
- *Business Process Management (BPM)*
- *Online Analytic Processing (OLAP)*
- *Data Mining*
- *Process Mining*

The answers to this question are summarized in the chart of Figure 7.3. The solid line represents the average domain familiarity for all respondents. Similarly, the dashed lines show the average domain familiarity for academics and industrials.

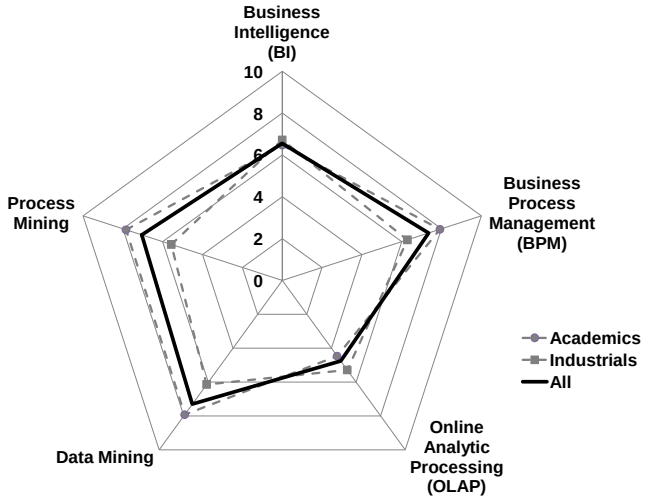


Figure 7.3: The domains of expertise of the workshop participants.

The results of Figure 7.3 show that the respondents are reasonably familiar with the subjects covered in the workshop experiment. On the one hand, industrials have a more homogeneous familiarity with all the covered subjects. On the other hand, academics are more familiar with process mining, BPM, and data mining, subjects which are part of their academic programs.

7.4 Efficiency

In this section, we present the results of the performance evaluation of the Multidimensional Process Explorer framework. This evaluation is based on execution data from the set of experiments described in Subsection 7.3.1. The discussion about the results of this evaluation is presented in Section 7.7.

Flexible Heuristics Miner

The evaluation of the Flexible Heuristics Miner consists of measuring the computational time of mining an aC-net from some given process data. Since this miner is a one-dimensional process discovery technique, the only aspect considered in this evaluation is the size of the dataset. Remark that this evaluation aims especially at the comparison of the efficiency aspect of the traditional and the multidimensional approaches. Therefore, the mining of long-distance dependencies (Algorithm 3) is not considered in these experiments once that this step does not exist in the multidimensional process discovery approach.

The computational complexity of the Flexible Heuristics Miner can be analyzed in two different aspects: the mining of the dependency graph (Algorithm 1), and the mining of the splits and joins (Algorithm 2).

Mining the dependency graph depends firstly on computing the dependency measures for all distinct activities appearing in the process data. The computational time required to compute this measuring information is quadratic ($O(n^2)$) to (a)

the number of distinct activities, and linear ($O(n)$) to (b) the total amount of process events. Similarly, the computational time required to build the dependency graph is quadratic ($O(n^2)$) to (c) the number of distinct activities, and linear ($O(n)$) to (d) the number of process events.

Mining the splits and joins consists basically of parsing process instances to find the activated inputs and outputs of activities (i.e., the splits and joins). The computational time required to compute splits and joins is quadratic ($O(n^2)$) to (e) the number of process events in the process instances (i.e., the instance length), and linear ($O(n)$) to (f) the number of process instances.

Presenting the mining time of the Flexible Heuristics Miner over datasets d_{1-6}, Figure 7.4 provides insight into the complexity characterization described above (items b, d, and f). The dark gray bars represent the computational time required to compute the splits and joints, while the light gray ones represent that for the dependency graph. The datasets are identified by number of process instances.

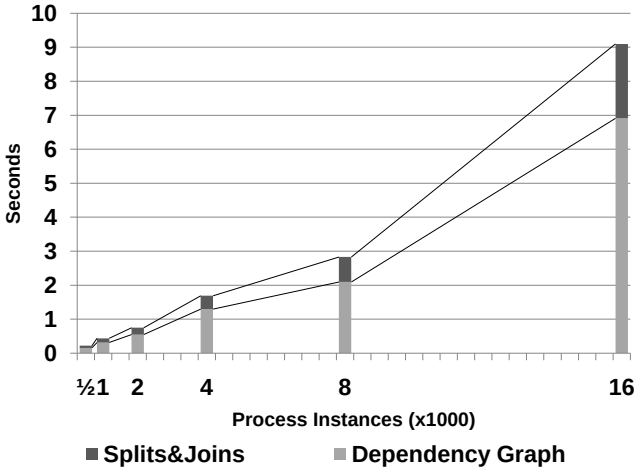


Figure 7.4: Mining time of an aC-net from datasets d_{1-6}.

Multidimensional Heuristics Miner

The evaluation of the Multidimensional Heuristics Miner consists of measuring the time required to mine mC-nets from some given process data. Three aspects are considered in this evaluation: the dimensionality of the data cube, the size of the dataset, and the cardinality of the dimensions. Remark that this evaluation assumes the existence of an Event Cube on which the Multidimensional Heuristics Miner can be executed. Any aspect related to Event Cube is not considered in this evaluation (e.g., the materialization time of the dependency measures).

The computational complexity of the Multidimensional Heuristics Miner can be analyzed in two different aspects: the mining of the multidimensional dependency graph (Algorithm 4), and the computing of the event occurrence bindings (Algorithm 5).

Mining the multidimensional dependency graph consists basically of retrieving the dependency measures from an Event Cube and building the dependency graph. The computational time required to retrieve – from an Event Cube – all

the necessary information to build the dependency graph is linear ($O(n)$) to (a) the number of distinct event occurrences, and constant ($O(1)$) to (b) the number of process instances and to (c) the number of process events. The computational time required to build the dependency graph is quadratic ($O(n^2)$) to (d) the number of distinct event occurrences, and linear ($O(n)$) to (e) the number of process events.

Computing the event occurrence bindings consists basically of parsing process instances to find the activated inputs and outputs of event occurrences (i.e., the input and output bindings that define the event occurrence bindings). The computational time required to compute event occurrence bindings is quadratic ($O(n^2)$) to (f) the number of process events in the process instances (i.e., the instance length), and linear ($O(n)$) to (g) the number of process instances.

Presenting the mining time of a mC-net with 1-D nodes over dimension *Activity* and 0-D edges, Figure 7.5 provides insight into the complexity characterization described above (items b, c, e, and g). The dark gray bars represent the computational time required to compute the event occurrences, while the light gray ones represent that for the multidimensional dependency graph. The datasets are identified by process instance amount.

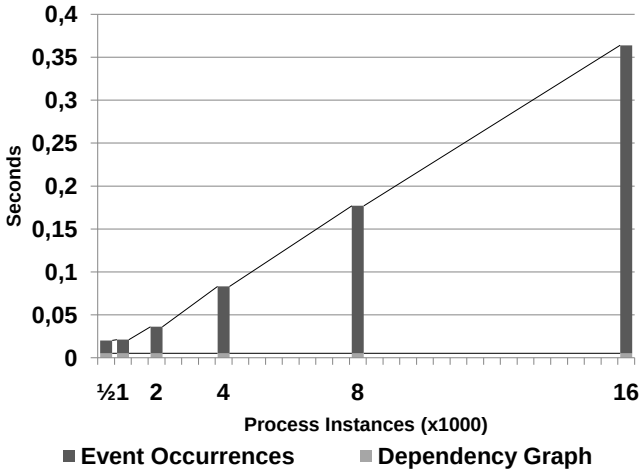
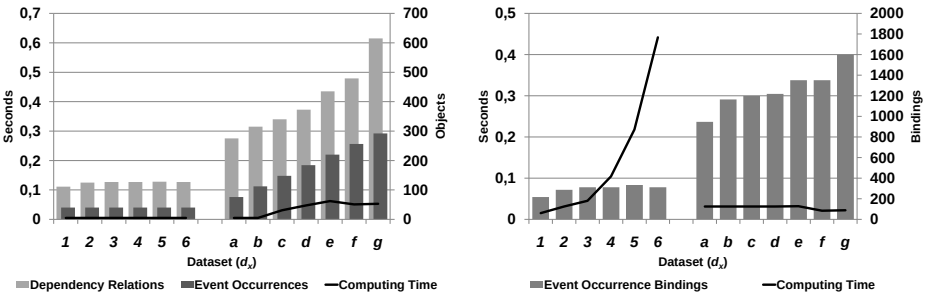


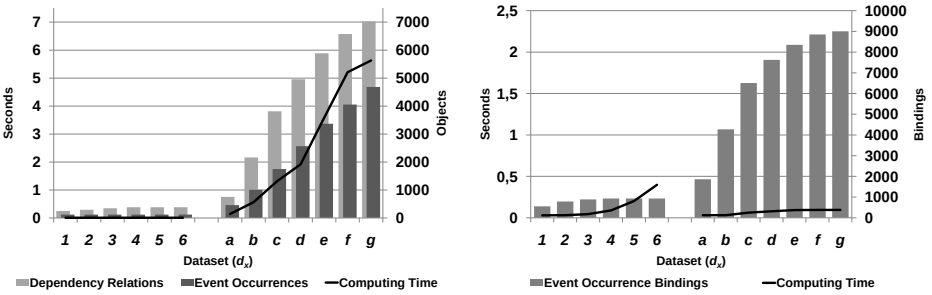
Figure 7.5: Mining time of a 1-D mC-net with 1-D nodes over dimension *Activity* and 0-D edges, from datasets d_{1-6}.

Further insight into the complexity characterization described above (items $a - e$ and g) is provided in figures 7.6 and 7.7. Figure 7.6 presents the mining time of a 2-D mC-net with 2-D nodes over dimension *Activity* and *Resource* and 0-D edges. Figure 7.7 presents the mining time of a 3-D mC-net with 2-D nodes over dimension *Activity* and *Resource* and 1-D edges over dimension *Product*. Figures 7.6a and 7.7a describe the computing time of the multidimensional dependency graph (line) as well as the amount of computed event occurrences (dark gray bars) and dependency relations (light gray bars). Similarly, Figures 7.6b and 7.7b describe the computing time of the event occurrence bindings (line) as well as the amount of computed bindings (bars).



(a) Mining the dependency graph. **(b)** Mining the event occurrence bindings.

Figure 7.6: Mining time of a 2-D mC-net with 2-D nodes over dimension *Activity* and *Resource* and 0-D edges.



(a) Mining the dependency graph. **(b)** Mining the event occurrence bindings.

Figure 7.7: Mining time of a 3-D mC-net with 2-D nodes over dimension *Activity* and *Resource* and 1-D edges over dimension *Product*.

Inverted Index

The evaluation of the Inverted Index consists of measuring the time required to index some given process data. In this evaluation, the indexing process is defined as:

i. **Transformation**: Deriving new time information (i.e., seven new time-based attributes) from the timestamps, as described in Subsection 5.2.2.

ii. **Indexation**: Indexing all attributes in the process data as well as the seven new time-based attributes. The list of the indexed attributes is provided in Table 7.2. In this table, A_{1-8} represent the attributes directly retrieved from the process data, while A_{4a-4g} the time-based attributes derived from A_4 (i.e., the timestamp). The cardinality of each attribute in the evaluating datasets (d_{1-6} and d_{a-g}) is also provided in the table.

iii. **Normalization**: Converting the event-based attributes *Product* and *Problem Type* into instance-based attributes, as described in Subsection 5.2.1.

The computational complexity of the indexing process can be characterized as follows.

- The computational time required to derive new information is linear ($O(n)$) to (a) the number of process events and to (b) the number of attributes to be derived.

ID	*Attribute*	*Cardinality*					
		d_1	d_2	d_3	d_4	d_5	d_6
A_1	*Case Identifier*	500	1000	2000	4000	8000	16000
A_2	*Activity*	12	12	12	12	12	12
A_3	*Resource*	13	13	13	13	13	13
A_4	*Timestamp*	1903	2902	5784	11415	22888	45122
A_5	*Product*	3	3	3	3	3	3
A_6	*Problem Type*	10	10	10	10	10	10
A_7	*Repair Number*	3	3	3	3	3	3
A_8	*Test Result*	2	2	2	2	2	2
A_{4a}	*Hour*	24	24	24	24	24	24
A_{4b}	*Date*	2	3	5	9	18	36
A_{4c}	*Day of Week*	2	3	5	7	7	7
A_{4d}	*Day of Month*	2	3	5	9	18	31
A_{4e}	*Month*	1	1	1	1	1	2
A_{4f}	*Quarter*	1	1	1	1	1	1
A_{4g}	*Year*	1	1	1	1	1	1

(a) Datasets d_{1-6}.

ID	*Attribute*	*Cardinality*						
		d_a	d_b	d_c	d_d	d_e	d_f	d_g
A_1	*Case Identifier*	1000	1000	1000	1000	1000	1000	1000
A_2	*Activity*	12	12	12	12	12	12	12
A_3	*Resource*	25	37	49	61	73	85	97
A_4	*Timestamp*	2252	2266	2260	2282	2270	2277	2256
A_5	*Product*	6	9	12	15	18	21	24
A_6	*Problem Type*	10	10	10	10	10	10	10
A_7	*Repair Number*	3	3	3	3	3	3	3
A_8	*Test Result*	2	2	2	2	2	2	2
A_{4a}	*Hour*	24	24	24	24	24	24	24
A_{4b}	*Date*	2	2	2	2	2	2	2
A_{4c}	*Day of Week*	2	2	2	2	2	2	2
A_{4d}	*Day of Month*	2	2	2	2	2	2	2
A_{4e}	*Month*	1	1	1	1	1	1	1
A_{4f}	*Quarter*	1	1	1	1	1	1	1
A_{4g}	*Year*	1	1	1	1	1	1	1

(b) Datasets d_{a-g}.

Table 7.2: List of the indexed attributes.

- The computational time required to index process data is linear ($O(n)$) to (c) the number of process instances, to (d) the number of process events, and to (e) the number of attributes to be indexed. Furthermore, this time is constant ($O(1)$) to (f) the number of distinct attributes values (i.e., the cardinality of the attribute).
- The computational time required to normalize event-based attributes is linear ($O(n)$) to (g) the number of process instances and to (h) the number of process events to be normalized. Furthermore, this time is constant ($O(1)$) to (i) the instance length.

Presenting the processing time of the indexing process of datasets d_{1-6}, Figure 7.8 provides insight into the complexity characterization described above (items $a - e$, g, and h). Similarly, presenting the processing time of the indexing process of datasets d_2 and d_{a-g}, Figure 7.9 provides insight into the items f and i. In both figures, the transformation, indexation, and normalization steps of the indexing process are identified by different colors. The datasets are identified by process instance amount.

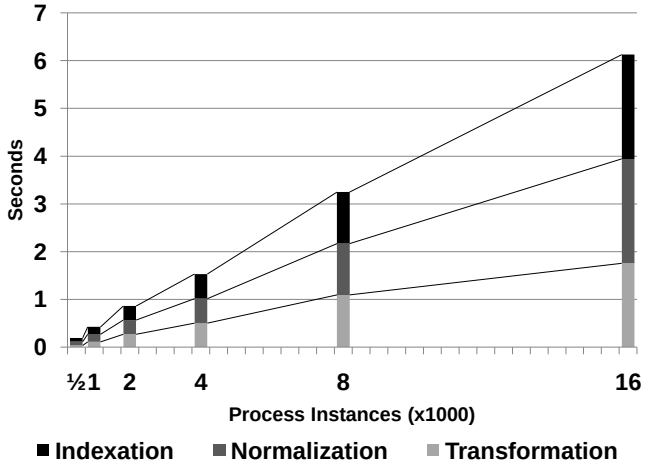


Figure 7.8: Processing time of the indexing process of datasets d_{1-6}.

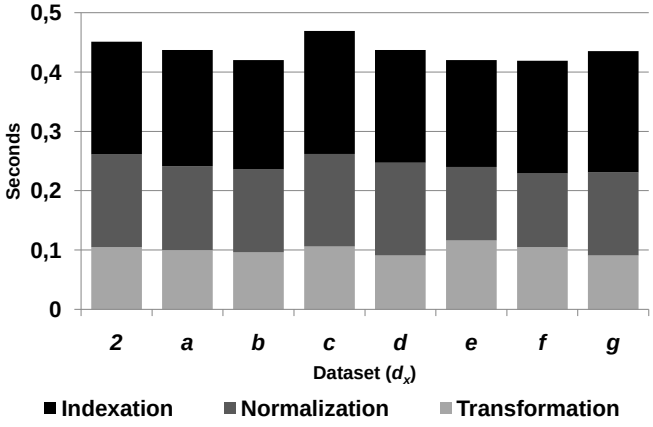


Figure 7.9: Processing time of the indexing process of datasets d_2 and d_{a-g}.

Event Cube

The evaluation of the Event Cube consists of measuring the time required to build (materialize) a data cube from some give process data. Materializing the control-flow and performance measures described in Table 7.3, three aspects are considered in this evaluation: the dimensionality of the data cube, the size of the dataset, and the cardinality of the dimensions.

ID	*Measure*	*Aggregation Function*
m_1	*Event Entry*	*Count*
m_2	*Event Start*	*Count*
m_3	*Event End*	*Count*
m_4	*Direct Successor*	*Count*
m_5	*Indirect Successor*	*Count*
m_6	*Length-Two Loop*	*Count*
m_7	*Direct Successor Dependency*	
m_8	*Indirect Successor Dependency*	
m_a	*Instance Entry*	*Count*
m_b	*Throughput Time*	*Average*

Table 7.3: Control-flow and performance measures used in the materialization process.

Four data cubes characterized by different number of dimensions are defined in order to assess the dimensionality aspect. A characterization of these data cubes is provided in Table 7.4. Table 7.4a identifies the dimensions of the data cubes, while Table 7.4b summarizes their number of cuboids and multidimensional cells.

n-D	*Dimensions*
1-D	Activity
2-D	Activity and Resource
3-D	Activity, Resource and Product
4-D	Activity, Resource, Product and Problem Type

(a) Dimensions of the data cubes.

n-D	*Cuboids*	*Multidimensional Cells*							
		d_{1-6}	d_a	d_b	d_c	d_d	d_e	d_f	d_g
1-D	2	13	13	13	13	13	13	13	13
2-D	4	182	338	494	650	806	962	1118	1274
3-D	8	728	2366	4940	8450	12896	18278	24596	31850
4-D	16	8008	26026	54340	92950	141856	201058	270556	350350

(b) Characteristics of the data cubes.

Table 7.4: Four possible data cubes with different dimensionality.

The computational complexity of the Event Cube can be analyzed in two different aspects: building the lattice of cuboids, and computing the measuring information of all multidimensional cells in the Event Cube.

Building the lattice of cuboids consists basically of computing all cuboids and corresponding multidimensional cells. The computational time required to compute all cuboids is exponential ($O(2^n)$) to (a) the dimensionality of the data cube. The same time required to compute the multidimensional cells of a specific cuboid is exponential ($O(2^n)$) to (b) the dimensionality of the cuboid, and linear ($O(n)$) to (c) the cardinality of the cuboid's dimensions.

Computing measuring information consists basically of computing the information related to specific measures, for all multidimensional cells in the Event Cube. The computational time required to compute this information is linear ($O(n)$) to (d) the number of non-empty multidimensional cells (i.e., cells that have support in the data), and to (e) the number of process instances. Furthermore, the computational time required to compute list- or set-based measures is linear ($O(n)$) to (f) the number of process events, while that for map-based measures (g) is quadratic ($O(n^2)$).

Presenting the materialization time of the data cubes described in Table 7.4 (datasets d_{1-6}), Figure 7.10 provides insight into the complexity characterization described above (items $a-c$ and e). The cube dimensionality is identified by different colors. The solid lines represent actual materialization time measurements, while the dotted ones consist of interpolated values. The datasets are identified by process instance amount.

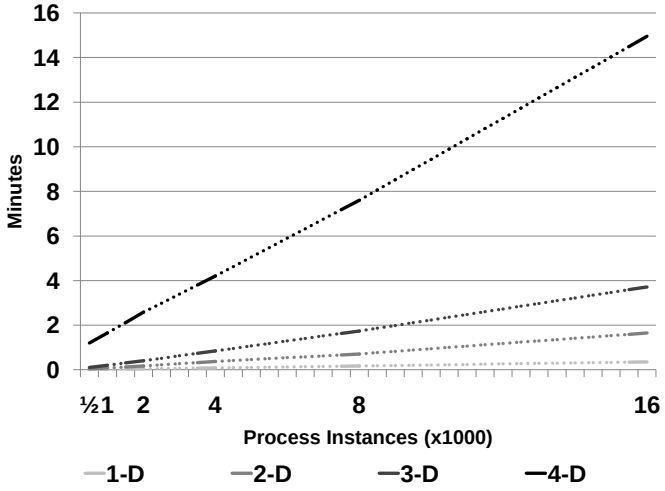
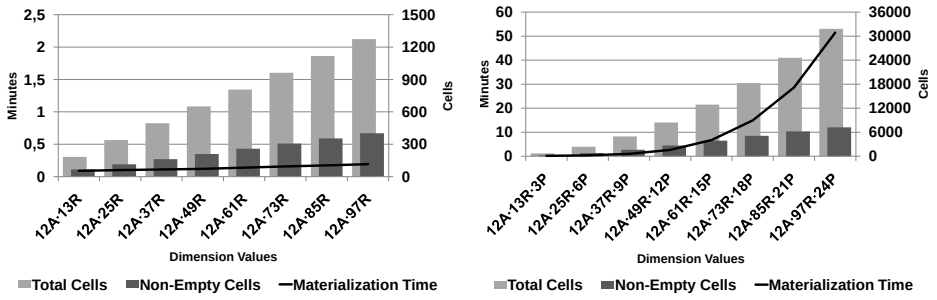


Figure 7.10: Materialization time of the data cubes described in Table 7.4, using datasets d_{1-6}.

Further insight into the complexity characterization described above (items $a-e$) is provided in Figure 7.11, which presents the materialization time of the 2-D and 3-D data cubes described in Table 7.4 (datasets d_2 to d_{a-g}). In this figure, the solid line represents the materialization time. The dark gray bars represent the number of non-empty multidimensional cells in the data cube, i.e., the cells that have support in the data. The light gray bars identify the total number of multidimensional cells in the data cube. The datasets are identified by amount of distinct dimension values considered in the data cube.

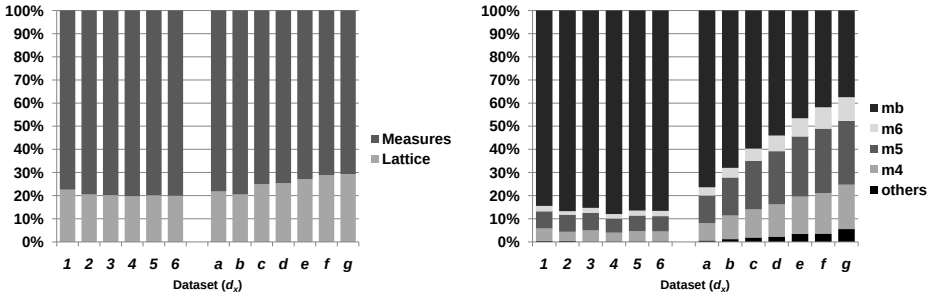
The materialization process can be characterized as a two-step process. In the first step, the lattice of cuboids of the data cube is built. This means that not only the



(a) 2-D data cube. **(b)** 3-D data cube.

Figure 7.11: Materialization time of the 2-D and the 3-D data cubes described in Table 7.4, using datasets d_2 and d_{a-g}.

structure of the data cube is built but also the cuboids and their multidimensional cells. In the second step, the measuring information of each multidimensional cell is computed with respect to a given set of control-flow and performance measures. Figure 7.12 provides insight into the complexity characterization described above (items f and g). Figure 7.12a presents the impact of these steps in the materialization process of the 2-D data cube described in Table 7.4. Figure 7.12b presents the impact of the measures considered in the materialization process (see Table 7.3) in the second step (i.e., the measure computation).



(a) The impact of building the lattice of cuboids and computing the measuring information in the materialization process.

(b) The impact of the measures in the process of computing the measuring information.

Figure 7.12: Characterization of the materialization time of the 2-D data cube described in Table 7.4.

Instance Clustering

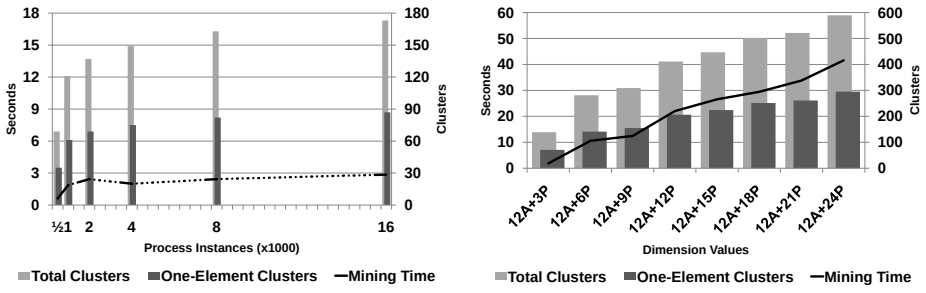
The evaluation of the Instance Clustering technique consists of measuring the time required to mine all instance clusters from a mC-net with 2-D nodes over dimensions *Activity* and *Product*, and 0-D edges. Two aspects are considered in this evaluation: the size of the dataset and the cardinality of the dimensions.

The computational complexity of the Instance Clustering technique can be analyzed in two different aspects: the computation of all distinct process instances, and the grouping of process instances according to instance similarity.

Computing all distinct process instances consists basically of transforming process instances into multisets of event occurrence bindings. The set of these multisets of bindings can be considered the set of distinct process instances. The computational time required to compute the multisets of bindings is linear ($O(n)$) to (a) the number of process instances and to (b) the number of process events.

Grouping process instances consists basically of computing the similarity between instances (or clusters of instances) and grouping the most similar cases iteratively until only one cluster remains. The computational time required to compute the similarity between instances (or clusters of instances) is quadratic ($O(n^2)$) to (c) the number of distinct process instances.[4] The computational time required to compute all instance clusters is linear ($O(n)$) to (d) the number of distinct process instances.[5]

Figure 7.13 provides insight into this complexity characterization (items $a - d$). Figure 7.13a presents, for datasets d_{1-6}, the mining time of all instance clusters as well as the amount of mined clusters. Similarly, Figure 7.13b presents the same information for datasets d_2 and d_{a-g}. The solid lines represent actual mining time measurements, while the dotted ones consist of interpolated values. The dark gray bars represent the number of one-element clusters mined from the mC-net, i.e., the distinct process instances that define the multidimensional process model. The light gray bars identify the total number of clusters mined from the mC-net. The datasets are identified either by process instance amount (Figure 7.13a) or by amount of distinct activities and products (Figure 7.13b).



(a) Datasets d_{1-6}. **(b)** Datasets d_2 and d_{a-g}.

Figure 7.13: Mining time of all instance clusters from a 2-D mC-net.

Frequent Binding Sets Miner

The evaluation of the Frequent Binding Sets Miner consists of measuring the time required to mine 1000 frequent binding sets from a mC-net with 1-D nodes over dimension *Resource* and 0-D edges. Two aspects are considered in this evaluation: the size of the dataset and the cardinality of the dimensions.

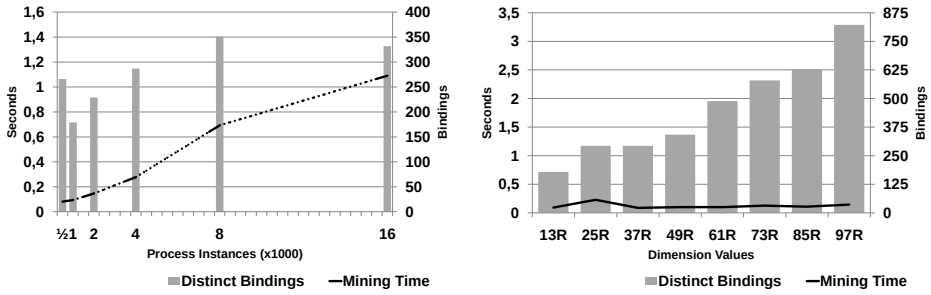
The computational complexity of the Frequent Binding Sets Miner technique can be analyzed in two different aspects: the candidate generation for frequent binding sets, and the computation of binding set supports.

[4] $(n-1)^2$ is the number of similarity computations needed to group n process instances.
[5] $2n-1$ is the number of distinct instance clusters that can be generated from n process instances.

Generating candidates for frequent binding sets consists basically of combining known frequent binding sets into new binding sets. The computational time required to compute all possible binding sets is exponential ($O(2^n)$) to (a) the number of distinct bindings.[6] Remark that this is the worst-case scenario. Typically, the candidate generation takes a threshold on the bindings set supports into account, which can lead to a significant reduction of the search space.

Computing the frequency of binding sets consists basically of intersecting sets of process event identifiers of bindings (or sets of bindings). The computational time required to compute the frequency of binding sets is linear ($O(n)$) to (b) the number of distinct bindings and to (c) the number of process events.

Figure 7.14 provides insight into this complexity characterization (items $a - c$). Figure 7.14a presents, for datasets d_{1-6}, the mining time of 1000 frequent sets as well as the amount of distinct event occurrence bindings from which the frequent sets are computed. Similarly, Figure 7.14b presents the same information for datasets d_2 and d_{a-g}. The solid lines represent actual mining time measurements, while the dotted ones consist of interpolated values. The bars represent the number of event occurrence bindings that define the mC-net. The datasets are identified either by process instance amount (Figure 7.14a) or by amount of distinct resources (Figure 7.14b).



(a) Datasets d_{1-6}. **(b)** Datasets d_2 and d_{a-g}.

Figure 7.14: Mining time of 1000 frequent binding sets from a 1-D mC-net.

7.5 Usability

In this section, we present the results of the usability evaluation of the Multidimensional Process Explorer framework. This evaluation is based on the feedback gathered in the workshop described in Subsection 7.3.3. Two usability aspects are considered in this evaluation: understandability and operability. The discussion about the results of this evaluation is presented in Section 7.7.

7.5.1 Understandability

In order to assess the understandability of the functionality provided by the Multidimensional Process Explorer framework, the following question was posed to the workshop participants: *In a scale of 1 (difficult) to 10 (easy), how easy are the results (i.e.,*

[6] $2^n - 1$ is the number of distinct binding sets that can be generated from n bindings.

the process analysis) to understand? The answers to this question are summarized in the box plot of Figure 7.15.

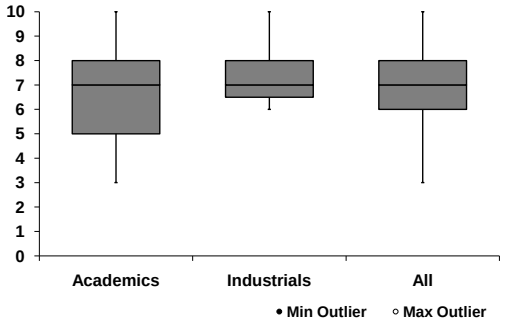


Figure 7.15: Feedback results about understandability. The boxes represent the interquartile range (IQR), which is the range between the 25^{th} percentile (Q1) and the 75^{th} percentile (Q3). The band in the box represents the 50^{th} percentile (Q2), i.e., the median. The vertical line represent the interval $[\text{Q1} - 1.5 \cdot \text{IQR}; \text{Q3} + 1.5 \cdot \text{IQR}]$. Any value outside this interval is considered as an outlier.

The results of Figure 7.15 show that the majority of the respondents consider that the framework provides understandable results. On the one hand, the feedback from the industrials is exclusively positive. On the other hand, the feedback from the academics is mainly positive but contains a few negative results (i.e., a few students consider the results not easy to understand). In general, it can be concluded that the results provided by the framework are understandable for the majority of users but not for all.

7.5.2 Operability

In order to assess the operability of the Multidimensional Process Explorer framework, the following question was posed to the workshop participants: *In a scale of 1 (difficult) to 10 (easy), how easy is the framework to use?* The answers to this question are summarized in the box plot of Figure 7.16.

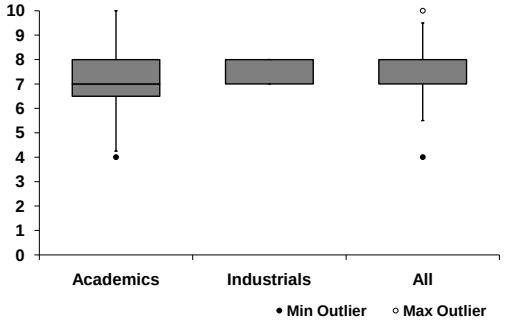


Figure 7.16: Feedback results about operability. The boxes represent the interquartile range (IQR), which is the range between the 25^{th} percentile (Q1) and the 75^{th} percentile (Q3). The band in the box represents the 50^{th} percentile (Q2), i.e., the median. The vertical line represent the interval $[\text{Q1} - 1.5 \cdot \text{IQR}; \text{Q3} + 1.5 \cdot \text{IQR}]$. Any value outside this interval is considered as an outlier.

The results of Figure 7.16 show that the majority of the respondents consider that the framework is easy to use but there still is some space for improvements. On the one

hand, the feedback from the industrials is quite homogeneous with the results varying from 7 to 8. On the other hand, the feedback from the academics is more broad with some extremely high results but also a few negative results (i.e., a few students consider the framework not easy to use). In general, it can be concluded that the framework is easy to operate and control.

7.6 Functionality

In this section, we present the results of the functionality evaluation of the Multidimensional Process Explorer framework. This evaluation is based on the results of the case study described in Subsection 7.3.2, and the feedback gathered in the workshop described in Subsection 7.3.3. Two functionality aspects are considered in this evaluation: suitability and accuracy. The discussion about the results of this evaluation is presented in Section 7.7.

7.6.1 Suitability

In order to assess the suitability of the Multidimensional Process Explorer framework, the following question was posed to the workshop participants (industrials): *Could this framework*

- *reduce the amount of work to solve specific problems in your business?*
- *solve problems that cannot be solved by any other tool in your business?*
- *help on discovering new problems in your business?*
- *be a useful tool for business monitoring?*
- *be a useful analytic tool for decision making?*
- *increase your process awareness?*

Remark that, this question was not posed to academics because it concerns the framework application in industry. The answers to this question are summarized in the chart of Figure 7.17. The solid line represents the amount of positive answers to a specific sub-question.

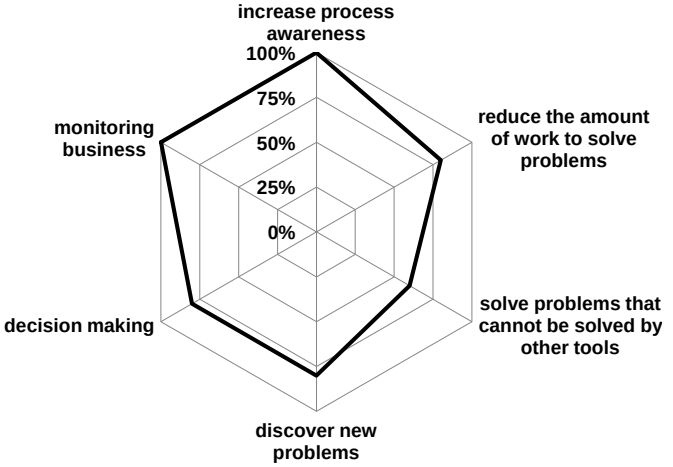


Figure 7.17: Feedback results about the suitability of the framework.

The results of Figure 7.17 show that the industrials totally agree that the framework could be used to improve the process awareness and to monitor the business. Additionally, they also agree that the framework could be a useful tool for decision making, discovering unknown problems, or even reducing the amount of work to solve problems. The respondents are more skeptical of using the framework to solve problems that cannot be solved by other tools. In general, it can be concluded that the framework provides appropriate functionality for the discovery and analysis of business processes.

In order to assess the suitability of the business process queries introduced in Subsection 5.3.2, the following question was asked to the workshop participants: *What combination of queries defines better your analysis needs?* Figure 7.18 summarizes the given answers to this question. The solid line represents the amount of positive answers to a specific query for all respondents. Similarly, the dashed lines show that amount for academics and industrials. Remark that the *how* query is not included in this evaluation because the current implementation of the framework cannot produce results without this query.

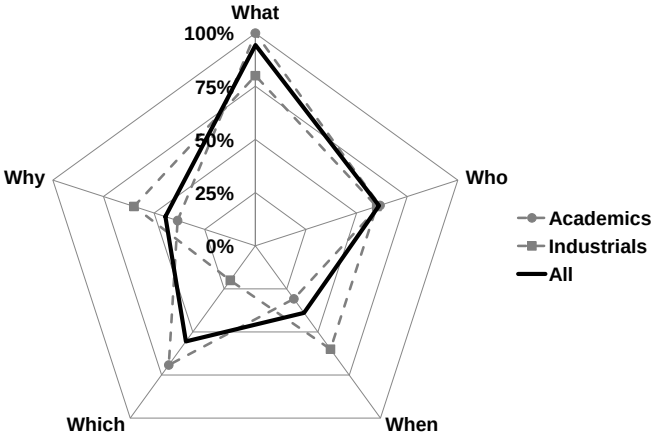


Figure 7.18: Feedback results about the suitability of business process queries.

The results of Figure 7.18 show that the respondents identify the *how + what* and *how + who* as the most suitable combinations of queries to their analysis needs. For the other queries, the results show that academics and industrials have distinct analysis needs. It is identified that *how + which* is also an interesting combination for academics, while *how + when* and *how + why* also suit the analysis needs of the industrials.

In order to assess the suitability of the knowledge discovery techniques described in Chapter 6, the following question was posed to the workshop participants: *What knowledge discovery techniques should be part of this framework?* The answers to this question are summarized in the chart of Figure 7.19. The solid line represents the amount of positive answers to a specific knowledge discovery technique for all respondents, while the dashed lines show that amount for academics and industrials.

The results of Figure 7.19 show that the respondents identify the *association rules* and the *classification* as the most suitable knowledge discovery techniques to their analysis needs. For the academics, the *clustering* is a technique that might be considered

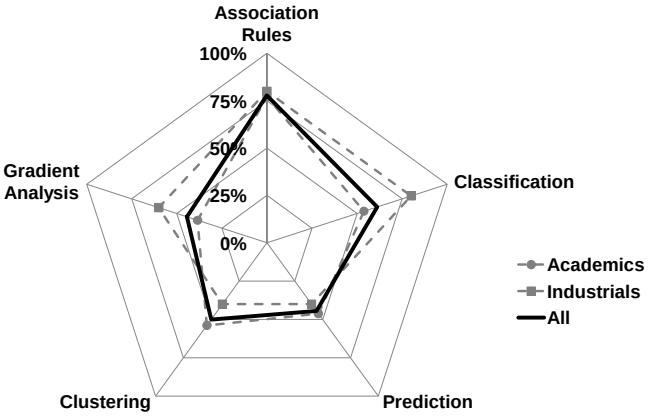


Figure 7.19: Feedback results about the suitability of knowledge discovery techniques.

as well. For the industrials, the *gradient analysis* is also identified as an interesting technique. Both academics and industrials consider the *prediction* as the less suitable knowledge discovery technique.

Complementing the workshop results, feedback about the suitability of the results of the Multidimensional Process Explorer framework was gathered in the case study. In general, business analysts consider that the multidimensional process models are the most suitable results. The characterization of process behavior for different aspects of the business process is identified as the functionality they would like to have available in their analyses. Furthermore, the comparison of process models (or sub-models) is a functionality the analysts believe that could produce interesting results. For example, the comparison of process behavior of support services performed in different countries. In contrast, business analysts consider that the process patterns do not bring new insight into the support services. This can be explained by the fact that the most frequent instances selected for analysis are characterized by simple process behavior (e.g., one or two tasks executed sequentially). Hence, similar insight into the performance of support services can be achieved by using existing analytical tools.

7.6.2 Accuracy

A simple evaluation of the accuracy of the Multidimensional Process Explorer framework was performed in the case study. Basically, this evaluation consisted of a discussion with five business analysts where some representative results of the explorative analysis were analyzed in terms of accuracy.

In general, the business analysts consider that the results provide a good and correct overview of the business processes in analysis. At a high level of abstraction, the generated process models describe the support services according to the analysts expectations. At lower levels of abstraction, it was not possible to derive any concrete conclusions once that results are too restrict. Regarding the accuracy of process patterns, the results suggest that measuring information provided in the patterns is accurate. All results describing unexpected behavior (or measuring information) were confirmed to be well-computed (i.e., the unexpected outcomes are supported in the data). Remark that the outcome of this discussion simply represents opinions based

on their experience. A further evaluation is still necessary to obtain a clear insight into the accuracy aspect of the Multidimensional Process Explorer framework.

7.7 Discussion

Three different analyses were conducted to evaluate a prototype that implements the multidimensional approach for discovery and analysis of business processes introduced in this thesis. The main focus of this evaluation concerns the feasibility and applicability of the approach as well as the efficiency aspects of the software.

Experiments

The results of the performance evaluation of the framework concerns the quantitative aspects of the functionality provided by the prototype, i.e., the efficiency characteristic of the software quality model. Using synthetic process data, the main framework's components were assessed in terms of time behavior in order to identify the system bottlenecks.

Process Discovery: The results suggest that the process discovery techniques introduced in this thesis are characterized by linear computational times to the amount of process instances and process events. An interesting observation is that almost all computational effort of building the dependency graph is related to the dependency measures computation. This explains the fact that the Multidimensional Heuristics Miner is capable of building a dependency graph in a fraction of time of the traditional approach once that the dependency measures are computed beforehand in the Event Cube. Another interesting observation is that the computation of splits and joins (or event occurrence bindings) supported by an inverted index is at least four times faster than the traditional approach (i.e., accessing directly the process data). Remark, however, that this observation does not take into account the time required to build the index. One of the conclusions of this evaluation is that, as claimed in this thesis, a multidimensional process model can indeed be generated on-the-fly using an Event Cube.

Process Analysis: The results confirm that the indexation process introduced in this thesis is characterized by linear computational times to the amount of process instances and process events. However, further experiments are needed to assess the performance of the inverted index with respect to retrieval operations. Regarding the Event Cube, the results suggest that the computational time required to materialize a data cube is (i) linear to the amount of process instances, process events, and non-empty multidimensional cells in the data cube, and (ii) exponential to the dimensionality of the data cube. The scalability of an Event Cube cannot be compared to traditional data cubes for some reasons. Apart implementation aspects and issues, an Event Cube is designed to hold several complex measures, which in this evaluation represent about three quarters of the materialization time. Nonetheless, the results show that the process discovery and analysis of business processes can be performed on low-dimensional Event Cubes. This limitation on dimensionality is not expected to have negative impact in application of the proposed approach. Discovery operations (e.g., drill-down

and roll-up) basically consist of navigating the lattice of cuboids, retrieving measuring information from the cuboids, and generating a mC-nets according to some process perspective. Therefore, the performance of these operations mainly and directly depends of the performance of the Multidimensional Heuristics Miner. Filtering operations (e.g., slice and dice) not only depends on the Multidimensional Heuristics Miner but also on the generation artificial dependency relations. Therefore, further experiments are needed to assess the performance of these operations.

Process Patterns: The results demonstrate that the mining of process patterns on an Event Cube is feasible. Two different mining techniques are implemented in the prototype: the Instance Clustering and the Frequent Binding Sets Miner. On the one hand, the computational time required to run the Instance Clustering technique seems to be linear to the number of distinct process instances. On the other hand, the computational time required to mine frequent binding sets seems to be linear to the number of process events.

The main conclusion of this performance evaluation is that the prototype provides a reasonable performance for process mining. Unlike traditional OLAP tools, the proposed approach is not designed to handle high-dimensional data cubes with millions of entries. This high-dimensional scenario would lead to an impractical process analysis because of the curse of dimensionality issue (i.e., almost every perspective would produce spaghetti models). Therefore, it is expected that the number of dimensions in the discovery and analysis of business processes using the proposed approach is not higher than five. This low-dimensionality scenario turns possible the execution of the proposed approach on desktop computers (instead of dedicated servers), especially if the amount of process data is not significantly high (up to a quarter of a million of process events).

Workshop

The results of the workshop experiment concerns the qualitative aspects of the functionality provided by the framework, i.e., the usability and functionality characteristics of the software quality model. In general, positive and encouraging feedback was received from the workshop participants.

As expected, understandability results show that the majority of the respondents consider that the framework provides understandable results. Since the multidimensional process models (mC-nets) adopt a similar representation as the traditional process models (aC-nets), it is expected that the results can be easily understood by anyone familiar with process models. It is also expected that some understandability issues represented in the results by some negative feedback are related to the curse of dimensionality.

Operability results show that the majority of the respondents consider that the framework is easy to use but there still is some space for improvements. As an OLAP-based tool, it is expected that the framework can be used intuitively by performing OLAP operations on an Event Cube. This expectation is confirmed by the results. However, it is identified that some functionalities of the framework's user interface should be easier to use (e.g., the filtering operations).

Suitability results show that the respondents consider that the framework provides appropriate functionality for the discovery and analysis of business processes. As men-

tioned before, the same kind of functionality could be performed by using traditional process mining techniques, but in a laborious and non-integrated way. Some feedback from the industrials confirms this fact. They consider that the strongest functionality of the framework is the integrated environment, especially in complex analysis. The results also characterize the analysis needs of the participants in terms of business process queries and process patterns. This feedback provides insight into what kind of functionality should be part of the framework.

Case Study

The results of the case study concern one of the qualitative aspects of the functionality provided by the framework, the functionality characteristic of the software quality model. These results consist basically of feedback from business analysts about the results of an explorative analysis using the Multidimensional Process Explorer framework.

Suitability results show that business analysts consider that the multidimensional process models are the most suitable results. Furthermore, although the functionality is not implemented, the comparative analysis of process models (or sub-models) is a functionality the analysts believe that could bring added value to their analyses. In contrast, business analysts consider that the process patterns do not bring new insight into the support services. This can be explained by the fact that the most frequent instances selected for analysis are characterized by simple process behavior (e.g., one or two tasks executed sequentially). Hence, similar insight into the performance of support services can be achieved by using existing analytical tools.

Accuracy results show that business analysts consider that the framework provides correct results. However, as stated before, this evaluation is based on feedback from a restrict number of analysts. Hence, the feedback about accuracy gathered in this case study should be seen simply as an indicator. A further evaluation is still necessary to obtain a clear insight into the accuracy of multidimensional process models and process patterns.

Conclusions

Two main conclusions can be drawn from the evaluation presented in this chapter. First, the feasibility of the multidimensional approach for discovery and analysis of business processes introduced in this thesis is demonstrated in an experimental study. Second, usability and functionality aspects of the Multidimensional Process Explorer framework are studied in a workshop experiment as well as a case study in industry. The gathered feedback indicates that the approach can provide new insight into business processes. Additionally, according to business analysts, the added value of the approach is the characterization of process behavior for different business perspectives. This cannot be achieved by any other tool.

Chapter 8

Conclusions

In this thesis, a multidimensional process discovery approach is proposed for knowledge discovery on process data. An overview of this approach is presented in Figure 8.1.

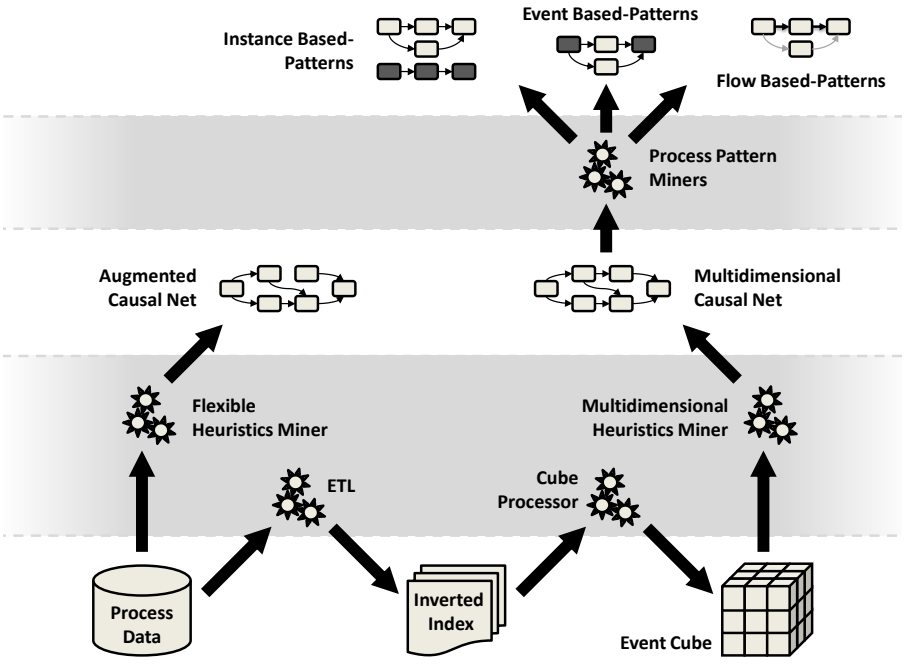


Figure 8.1: Overview of the multidimensional process discovery approach. Layers in gray color characterize data transformations, while the others identify process data or results. Arrows represent the data flow.

The multidimensional process discovery approach presented in this thesis consists of a sequence of steps in which process data are transformed first into process information, and then into process knowledge. In traditional process discovery approaches, process data are analyzed directly in order to get insight into the behavior of the business process described in the data. The Flexible Heuristics Miner is introduced as a traditional process discovery application in which process data can be transformed into process

information (i.e., a process model). The insight gained from the analysis of process information can be considered as process knowledge. In the multidimensional process discovery approach, process data are firstly structured to enable the data exploitation according to different business perspectives. The inverted index and the Event Cube are introduced to facilitate the dynamic exploitation of process data, which can be performed by a multidimensional process discovery application. As a generalization of the Flexible Heuristics Miner, the Multidimensional Heuristics Miner is introduced to exploit the different business perspectives by also transforming process data into – multidimensional – process information (i.e., a multidimensional process model). Process knowledge can then be achieved either by analyzing multidimensional process information or by extracting non-trivial patterns from that information. Several techniques for extracting process patterns from a multidimensional process model are introduced in this thesis.

A discussion covering the research questions investigated in this thesis is provided next. Then, the contributions, applications, and limitations of the multidimensional process discovery approach are discussed in sections 8.1, 8.2, and 8.3. Finally, this thesis is concluded by presenting future work directions (Section 8.4).

Research Questions

The research work presented in this thesis is defined by one main research question (Q), and six related sub-questions (q_{1-6}). A final discussion about these research questions is provided next.

(q_1) *Which traditional process mining techniques can be extended with multidimensional capacity (i.e., be applied using different dimensions, by turning traditional analyses into multidimensional)?*

An overview of the process mining domain is presented in Chapter 1. *Discovery*, *conformance*, and *enhancement* are identified as the main applications of process mining. From these, discovery is identified as the most suitable application once that it simply relies on process data to be applied. Enhancement can also be considered for extending the process discovery results with performance information. Under the process discovery scope, four perspectives are identified: *control-flow*, *organization*, *time*, and *data*. The different aspects (dimensions) of the business process are described in these perspectives. Therefore, by integrating dynamically all of these perspectives, it is possible to extend the process discovery with multidimensional capacity.

(q_2) *How can traditional process discovery be extended with multidimensional capacity?*

The link between traditional and multidimensional process discovery and analysis is established in chapters 3 and 4. Generalizing traditional process models, multidimensional process models are defined to represent the different process discovery perspectives (i.e., *control-flow*, *organization*, *time*, and *data*). Multidimensional capacity is achieved by combining process discovery perspectives.

(q_3) *What kind of knowledge can be achieved with dynamic multidimensional process discovery and analysis?*

As described in Chapter 5, business process queries can be used to characterize the different kinds of analyses that can be performed by process discovery techniques. By

integrating the *organization*, *time*, and *data* perspectives into the *control-flow* one, dynamic multidimensional process discovery and analysis can provide the same insight into a business process as those process discovery perspectives, but in an integrated environment, on-the-fly, and at different levels of abstraction.

(q_4) *How can multidimensional process discovery results be exploited for knowledge discovery?*

As described in Chapter 5, the Event Cube organizes the process data according to multiple business perspectives. Each business perspective is characterized by a distinct set of dimensions and can provide insight into specific aspects of a business process. Knowledge discovery on an Event Cube can be achieved by exploiting these perspectives. Based on main OLAP operators, discovery and filtering operations are introduced to facilitate hypothesis-driven analysis of business processes (cf. Section 5.3). Based on data mining functionalities, techniques for automatic extraction of non-trivial process patterns are also proposed (cf. Chapter 6).

(q_5) *What kind of non-trivial patterns can be – automatically – derived from multidimensional process discovery results?*

Process patterns as described in Chapter 6 are the answer to this sub-question. Instance-, event-, and flow-based patterns are possible patterns that can be derived from a multidimensional process model.

(q_6) *What is the added value of dynamic multidimensional process discovery and analysis to the stakeholders?*

This sub-question is partially answered in Chapter 7. A workshop and a case study were conducted to collect feedback from industrials and academics about the *functionality* and the *usability* of a prototype implementing a framework for dynamic multidimensional process discovery and analysis. Additionally, some experiments were conducted to get insight into the *efficiency* aspect of the framework. The results suggest that dynamic multidimensional process discovery and analysis can play an important role in understanding and monitoring business processes in enterprises, improving thus the process awareness of stakeholders. However, due to the limited scope of the current evaluation, it is not possible to provide a complete answer to this sub-question. A further evaluation where the framework is used by process analysts in real-life applications is still needed.

(Q) *What is the impact – and the benefit – of the dynamic multidimensional discovery and analysis of business processes?*

The dynamic multidimensional discovery and analysis of business processes is discussed in the core chapters of the thesis. The potential of this approach is the dynamic discovery and analysis of business processes according to specific perspectives, in an integrated environment and at different levels of abstraction. Therefore, using hypothesis-driven analysis, different aspects of business processes can be taken into account in the process discovery and analysis, which enables the process analyst to explore dynamically the business process according specific requirements. As stated before, this could eventually be done using traditional process mining techniques by filtering and transforming the process data, but in a laborious, time-consuming and non-integrated way. For instance, comparing the process behavior of the repair of 10 different products over

the last 12 months would require the partition of the process data into 120 datasets (10 products × 12 months). After partitioning the process data, a control-flow mining algorithm (e.g., the Flexible Heuristics Miner) needed to be applied on each of these datasets. As result, 120 process models would be generated separately. Comparing the process behavior represented in all of these process models only could be performed by model matching. All of these steps would take several hours to be achieved. Using the approach proposed in this thesis, the process behavior characterized by product and month would be generated on-the-fly from the process data, and the results would be presented in a single – multidimensional – process model.

Besides the research questions discussed above, a preliminary research question was investigated to position the main focus of this thesis (the knowledge discovery on process data) with respect to the main goal of the DataFusion project (the integration of information from the existing NPD datasets).

(Q') *How to establish an effective connection between the – operational – data sources and the knowledge discovery techniques?*

This research question is addressed in Chapter 2. Conceptually, a data provider can be used to establish an effective connection between the data sources and the knowledge discovery techniques (e.g., the framework for multidimensional process discovery described in this thesis). Two different approaches were identified as data providers: the data warehousing systems and the enterprise search engines. The first one can be seen as the logical option once that this kind of system is optimized to integrate data from heterogeneous sources for decision supporting. However, since NPD processes are dynamic and multidisciplinary, some of the prerequisites for implementing a data warehousing system may not be met. Hence, as an alternative approach, the search engine systems are considered for overcoming these issues.

8.1 Contributions

The contributions of this thesis to the data mining, OLAP, and process mining domains can be summarized as follows.

Process Representation

Defined in Chapter 3, the **Causal nets** [117] are designed to provide a simple and clear overview of processes mined from process data. Extending this process modeling formalism, **augmented Causal nets** are defined to complement the process model with measuring information (e.g., frequency) about activities, dependency relations, and input and output bindings (i.e., joins and splits). By characterizing the elements of process models (i.e., the process events and dependency relations) through multiple dimensions, **multidimensional process models** can be considered as a generalization of traditional (one-dimensional) process models. While the representation of a traditional process model consists of a directed graph, a multidimensional process model is represented by a directed multigraph. Generalizing augmented Causal nets, **multidimensional Causal nets** are defined as an instance of this type of process model.

Discussed in Chapter 4, the similarity assessment on multidimensional process models can be performed by using **similarity functions** [16, 35]. These functions are defined to compute the similarity between elements of multidimensional

process models, i.e., (i) event occurrences, (ii) dependency relations, (iii) event occurrence bindings, or (iv) sets of event occurrence bindings.

Control-Flow Mining

Presented in Chapter 3, the **Flexible Heuristics Miner** consists of an extension of Heuristics Miner algorithm [131, 132] where a new method for computing splits and joins is introduced. This control-flow algorithm produces an augmented Causal net representing the process described in some given process data. As an adaptation of the Flexible Heuristics Miner, the **Multidimensional Heuristics Miner** produces a multidimensional Causal net characterized by multiple aspects (dimensions) of the business process. The resulting Causal net represents a specific perspective of the process described in some given process data.

Process Discovery and Analysis

Presented in Chapter 5, the **Event Cube** (a data cube of process events [29, 50, 73]) is designed to organize summarizing information about different perspectives of a process described in some given process data. Summarizing information consists of **control-flow measures** and **performance measures**. Control-flow measures facilitate the execution on-the-fly of a control-flow mining algorithm (e.g., the Multidimensional Heuristics Miner) on a specific business perspective. Performance measures can be used to enhance the multidimensional process model with measuring information. An Event Cube relies on an **inverted index** (a data structure that facilitates the direct access to data characterized by specific constraints [145]) and can be considered as a framework for multidimensional discovery and analysis of business processes.

The multidimensional discovery and analysis of business processes can be achieved by exploiting an Event Cube. Based on typical OLAP operators [29, 50], **discovery operations** and **filtering operations** are introduced to facilitate the Event Cube exploitation. Discovery operations can be used to characterize the process perspective in analysis in terms of dimensions. As result, a multidimensional process model representing the perspective is generated on-the-fly. Filtering operations can be used to omit irrelevant information from the process model in analysis. A **filtering methodology** is introduced to ensure process behavior consistency. Challenges related to the multidimensional discovery and analysis of business processes are raised and discussed in this thesis.

Process Patterns

Discussed in Chapter 6, the extraction of process patterns can be achieved by applying some data mining functionalities on multidimensional process models. *Summarization*, *deviation analysis*, *rule learning*, *classification*, and *clustering* are presented as potential functionalities [40, 50]. Relying on these functionalities, **frequent process patterns**, **process gradients**, and – process – **instance clustering** are introduced as examples of techniques for extraction of process patterns.

The contributions of this thesis to the enterprise information systems, information retrieval, and NPD domains can be summarized as follows.

Concept architecture

Introduced in Chapter 2, this architecture consists of an enterprise search engine [53] capable of

i. searching and retrieving data from heterogeneous systems,

ii. transforming data according specific requirements,

iii. performing knowledge discovery, and

iv. storing knowledge and sharing it within the organization.

According to the DataFusion project requirements, this architecture is designed to establish an effective connection between the data sources and the knowledge discovery techniques. The framework for multidimensional discovery and analysis of business processes described in this thesis is identified as part of this architecture.

Knowledge discovery on process data
The knowledge discovery on process data according to the DataFusion project requirements is demonstrated in this thesis through two case studies.

8.2 Applications

Developed under the scope of the DataFusion project, the framework for multidimensional discovery and analysis of business processes introduced in this thesis was required to apply knowledge discovery techniques on New Product Development (NPD) datasets. A case study conducted in industry demonstrated that NPD – especially the customer support service – is a possible application for the proposed multidimensional approach. Other industry-related applications are also possible.

As any other process discovery technique, the multidimensional process discovery approach described in this thesis can be applied on data describing any kind of business process. Concrete process-related applications of the approach can be described as follows.

Control-flow mining can be applied to discover a – traditional or multidimensional – process model that reflects the process behavior observed in process data.

Hypothesis-driven process analysis consists of exploring the different business perspectives in order to discover and analyze the corresponding processes.

Temporal analysis can be performed by considering any time-related dimension. This type of analysis can provide insight into differences in process behavior that happen at different points in time.

Organizational analysis can be performed by considering any resource-related dimension. The *handover of work* is the most straightforward resource-related analysis in this multidimensional approach.

Decision point analysis consists of characterizing decisions in process models. In this multidimensional approach, this type of analysis can be performed either by applying classification techniques on multidimensional process models or by hypothesis-driven process analysis.

Performance analysis can be achieved by enhancing the multidimensional process models with performance information.

Sequence analysis can be performed by characterizing process instances according to some performance measure. In this multidimensional approach, process behavior (e.g., parallel behavior) is taken into account to group similar process instances.

Process pattern extraction consists of applying data mining functionalities on multidimensional process models in order to find non-trivial patterns. These patterns can provide further insight into the process behavior represented in the process model.

8.3 Limitations

The main limitations of the multidimensional process discovery approach can be described as follows.

Results consistency can be considered one of the main limitations of the multidimensional process discovery approach. Described in Subsection 5.4.4, this issue describes the difficulty to generate consistent process models across business perspectives. Although this issue can be minimized by user intervention, alternative strategies for exploiting Event Cubes (to the one presented in Chapter 5) should be developed to ensure the results consistency. For example, the solution introduced in Appendix A.

Result quality assessment is not considered in the proposed approach. Therefore, information about the accuracy of both multidimensional process models and process patters cannot be provided to the user at any moment in time. Considered also as a main limitation of the multidimensional process discovery approach, this issue may be solved by applying conformance and validation techniques on the approach results.

Curse of dimensionality is a known issue in the multidimensional process discovery approach. Discussed in Subsection 5.4.1, this issue describes the excessive complexity of process models by representing too much information. Although some approaches are proposed in this thesis to minimize the complexity of process models (e.g., using simplified model representations or applying filtering operations), the curse of dimensionality issue is still considered a limitation of the multidimensional process discovery approach.

8.4 Future Work

Besides the limitations and issues discussed in the previous section, five further topics are identified as future work.

Discovery-driven exploration of Event Cubes

Currently, the hypothesis-driven exploration is the only approach considered to explore Event Cubes. Basically, this approach relies exclusively on the user input to characterize the business perspective to be analyzed. A different exploration approach, the discovery-driven exploration [99], can be considered to provide support to the user on the Event Cube exploration. In this approach, potentially interesting information (e.g., exceptions or outliers) can be computed as measures, guiding the user in the process analysis of every business perspective.

Applying other process discovery techniques on Event Cubes

Currently, the Multidimensional Heuristics Miner is the only process discovery technique applied on the Event Cube. Nonetheless, other techniques can also be applied to gain further insight into different aspects of the business process. For

example, social network analysis may be performed on any process perspective containing information about resources.

Visualization of multidimensional process models

New forms of visualizing multidimensional process models can be studied. Currently, multidimensional process models are represented as directed multigraphs, adopting almost the same representation as traditional process models. These representations have proven to be relatively effective on describing traditional – and low-dimensional – process models. Nonetheless, different representations may also be considered to highlight specific aspects of the process.

Further discovery and filtering operations

New discovery and filtering operations can be developed to improve the Event Cube exploitation. For example, new discovery operations can focus simply on specific process elements, leaving the remaining elements unaltered. Drilling down simply an event occurrence (instead of the entire process model) may improve the process analysis by adding a limited amount of information.

Further process pattern extraction techniques and strategies

New process pattern extraction techniques can be developed to gain insight into different aspects of multidimensional process models. For example, graph mining techniques [50] can be applied on the process model structure to identify the most relevant nodes and edges in the graph. Another interesting possibility is the application of process pattern extraction techniques across business perspectives (e.g., comparing process behavior in models from different perspectives).

Appendix A

Building Multidimensional Process Models from a Reference Process Model

Let R be a multidimensional process model given as reference model and I the index over R that identifies whether a pair of identifiers is associated to a dependency relation in R. A multidimensional process model M can be built from R by:

i. Determining the set of event occurrences in M.

ii. Generating the set of candidate dependency relations in M. For each candidate dependency relations, the set of pair of identifiers that support the candidate relation should be computed as well.

iii. Checking, for each candidate dependency relation, whether at least a pair of identifiers that supports the candidate relation exists in I. If so, the candidate dependency relation is added to the dependency graph of M.

iv. Computing the event occurrence bindings of M using Algorithm 5. Note that this step does not take into account any information of R.

The following example is presented to illustrate how a multidimensional process model can be derived from another model. Let's consider that the process data consist of five process instances. The characteristics of these process instances are presented in Figure A.1.

Each process event is characterized by two attributes: *Activity* and *Type*. Additionally, an event identifier is also provided (the label in brackets). This information can be summarized as follows.

- $\{Activity{:}A\}$: $\{a, e, k, o, s\}$
- $\{Activity{:}B\}$: $\{b, c, f, g, h, i, l, m, p, q, t, u, v, w\}$
- $\{Activity{:}C\}$: $\{d, j, n, r, x\}$
- $\{Type{:}Start\}$: $\{b, f, h, l, p, t, v\}$
- $\{Type{:}Complete\}$: $\{a, d, e, g, i, j, k, m, n, o, q, r, s, u, w, x\}$

Figure A.2 presents the process model with 1-D nodes over dimension *Activity* and 0-D edges to be used as reference. The extra information provided in nodes

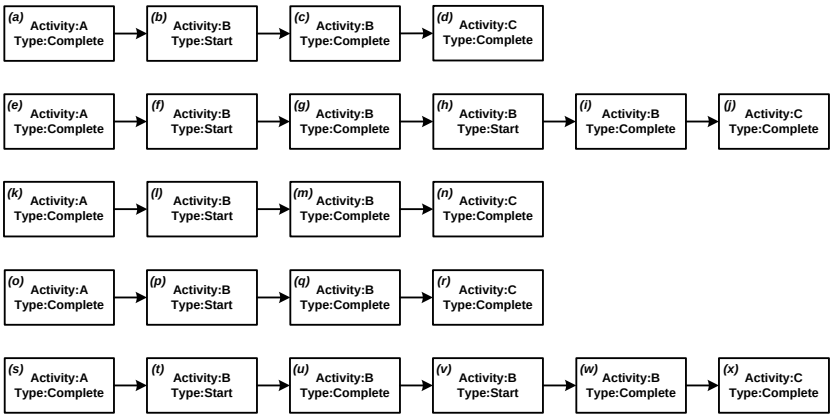


Figure A.1: Five possible process instances of a random business process.

represent the set of event identifiers that define the event occurrence. Analogously, that information on edges identify the causal dependencies in terms of process events.

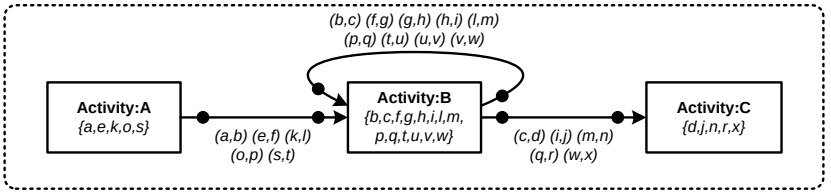


Figure A.2: The reference process model.

By indexing the information about the causal dependencies of the reference process model, it is possible to derive any other multidimensional process model of the same business process. For instance, the process model with 1-D nodes over dimension $Type$ and 0-D edges can be generated as follows.

i. First, the set of event occurrences to be represented in the process model should be identified. Using the summarized information about event identifiers, this set can be easily defined as $E = \{\ \{Type{:}Start\}, \{Type{:}Complete\}\ \}$.

ii. Second, the set of candidate dependency relations is defined as

$$D = \{(x, y) \mid \exists_{a,b \in E}[x = a \wedge y = b]\}.$$

As the dependency relations of the reference process model, each dependency relation in D can be characterized by a set of pairs of identifiers. For a given candidate dependency relation (x,y), this set can be defined as

$$T_{x \to y} = \{(i, j) \mid \exists_{a \in T_x, b \in T_y}[a \neq b \wedge i = a \wedge j = b]\},$$

where T_x and T_y are the sets of event identifiers of x and y.
In the running example, the candidate dependency relations are:

1. $(\{Type{:}Start\}, \{Type{:}Start\})$,

2. $(\{Type{:}Start\}, \{Type{:}Complete\})$,
3. $(\{Type{:}Complete\}, \{Type{:}Start\})$, and
4. $(\{Type{:}Complete\}, \{Type{:}Complete\})$.

Considering $T_{\{Type:Start\}} = \{a, e, k, o, s\}$ as the set of event identifiers of $\{Type{:}Start\}$, the set of pairs of identifiers of the first candidate is defined as $T_{\{Type:Start\}\rightarrow\{Type:Start\}} = \{(a, e), (a, k), (a, o), ..., (s, k), (s, o)\}$.

iii. Next, the dependency relations to be added to the dependency graph are selected from the candidate dependency relations. This selection can be defined as follows. Let I be the set of pairs of identifiers of the dependency relations of the reference process model. A given candidate dependency relation $(x, y) \in D$ is selected to be added to the dependency graph if I contains at least one element of $T_{x\rightarrow y}$.
For the running example, Figure A.3 presents the multidimensional dependency graph with the selected dependency relations. Like in the reference process model, the extra information represents the object's identifiers.

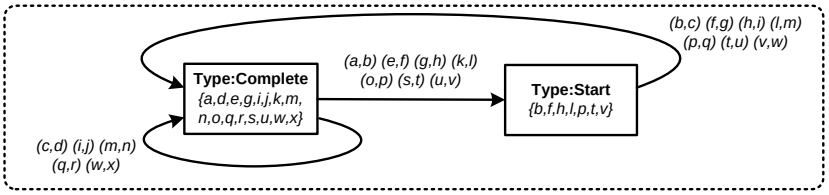


Figure A.3: The multidimensional dependency graph of the derived process model.

iv. Finally, the event occurrence bindings of the new process model can be computed using Algorithm 5.

Remark that no constrained dependency relation was considered in this example. Nonetheless, this issue can easily be overcome by applying the same approach as in Algorithm 4. First, both event and workflow constraints are combined to characterize exclusively the event occurrences. Then, based of these event occurrences, the dependency relations are computed without any constraint. At the end, the event occurrences and dependency relations can be normalized by applying the function *split*() (cf. Algorithm 4).

Bibliography

[1] ISO/IEC 9126-1:2001 - Software Engineering – Product Quality – Part 1: Quality Model, 2001. (cited on p. 169)

[2] A.K. A. de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2006. (cited on p. 10)

[3] A.K. A. de Medeiros and C.W. Günther. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005. (cited on p. 175)

[4] A.K. A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow Mining: Current Status and Future Directions. In R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20498-5. (cited on p. 10)

[5] A.K. A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Quantifying process equivalence based on observed behavior. *Data and Knowledge Engineering*, 64 (1):55–74, January 2008. ISSN 0169-023X. (cited on p. 87)

[6] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In *Proceedings of the 6th Workshop on Business Process Intelligence (BPI 2010)*, 2010. (cited on pp. 9, 11, and 43)

[7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc. (cited on pp. 5 and 145)

[8] R. Agrawal, T. Imieliński, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. ACM. ISBN 0-89791-592-5. (cited on p. 5)

[9] R. Alves, O. Belo, and J.T.S. Ribeiro. Mining top-k multidimensional gradients. In *Proceedings of the 9th International Conference on Data Warehousing and Knowledge Discovery*, pages 375–384, Berlin, Heidelberg, 2007. Springer-Verlag. (cited on p. 149)

[10] R. Alves, J.T.S. Ribeiro, and O. Belo. Mining significant change patterns in multidimensional spaces. *International Journal of Business Intelligence and Data Mining*, 4 (3/4):219–241, November 2009. ISSN 1743-8195. (cited on p. 8)

[11] R. Alves, J.T.S. Ribeiro, O. Belo, and J. Han. Ranking gradients in multi-dimensional spaces. *Complex Data Warehousing and Knowledge Discovery for Advanced Retrieval Development: Innovative Methods and Applications. Hershey, PA, EUA: IGI Global*, pages 251–269, 2009. (cited on p. 8)

[12] S. Angelov, P.W.P.J. Grefen, and D. Greefhorst. A framework for analysis and design of software reference architectures. *Information and Software Technology*, 54(4):417–431, April 2012. ISSN 0950-5849. (cited on pp. 23 and 24)

[13] M. Ankerst, M.M. Breunig, H.P. Kriegel, and J. Sander. OPTICS: Ordering Points to Identify the Clustering Structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 49–60, New York, NY, USA, 1999. ACM. ISBN 1-58113-084-8. (cited on p. 7)

[14] E. Badouel and P. Darondeau. Theory of Regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586. Springer Berlin / Heidelberg, 1998. ISBN 978-3-540-65306-6. (cited on p. 10)

[15] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, Second Edition*. Addison-Wesley Professional, April 2003. ISBN 0321154959. (cited on p. 23)

[16] M. Becker and R. Laue. A comparative survey of business process similarity measures. *Computers in Industry*, 63(2):148–167, 2012. ISSN 0166-3615. (cited on pp. 87 and 200)

[17] I.E. Ben-Gal. Outlier detection. In *The Data Mining and Knowledge Discovery Handbook*, pages 131–146. Springer, 2005. (cited on p. 103)

[18] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, pages 375–383, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75182-3. (cited on p. 10)

[19] S. Betz, D. Eichhorn, S. Hickl, S. Klink, A. Koschmider, Y. Li, A. Oberweis, and R. Trunko. 3D representation of business process models. In *MobIS'08*, pages 73–87, 2008. (cited on p. 16)

[20] K. Beyer and R. Ramakrishnan. Bottom-Up Computation of Sparse and Iceberg CUBEs. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, pages 359–370, New York, NY, USA, 1999. ACM. ISBN 1-58113-084-8. (cited on p. 8)

[21] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 0-89791-497-X. (cited on p. 6)

[22] C. Bratosin, N. Sidorova, and W.M.P. van der Aalst. Distributed genetic process mining. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010*, pages 1–8. IEEE, 2010. (cited on p. 10)

[23] C. Bratosin, N. Sidorova, and W.M.P. van der Aalst. Distributed Genetic Process Mining Using Sampling. In V. Malyshkin, editor, *Parallel Computing Technologies*, volume 6873 of *Lecture Notes in Computer Science*, pages 224–237. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-23177-3. (cited on p. 10)

[24] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984. (cited on p. 6)

[25] A.C. Brombacher, E.E. Hopma, R.A. Ittoo, Y. Lu, I.M. Luyk, L. Maruster, J.T.S. Ribeiro, A.J.M.M. Weijters, and J.C. Wortmann. Improving product quality and reliability with customer experience data. *Quality and Reliability Engineering International*, 2011. ISSN 1099-1638. (cited on p. 12)

[26] P.R. Carlile. Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge Across Boundaries. *Organization Science*, 15(5):555–568, October 2004. ISSN 1526-5455. (cited on p. 4)

[27] J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 358–373, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85757-0. (cited on p. 10)

[28] A.V. Castro. The Customer Service in Philips Cardio/Vascular Unit: Modeling and Improvement Strategies for the Current Process. Master's thesis, Eindhoven University of Technology, Eindhoven, 2009. (cited on p. 29)

[29] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26:65–74, March 1997. ISSN 0163-5808. (cited on pp. 7 and 201)

[30] C. Chen, X. Yan, F. Zhu, J. Han, and P.S. Yu. Graph OLAP: A multi-dimensional framework for graph data analysis. *Knowledge and Information Systems*, 21:41–63, October 2009. ISSN 0219-1377. (cited on p. 8)

[31] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing Petri Nets from State-Based Models. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '95, pages 164–171, Washington, DC, USA, 1995. IEEE Computer Society. ISBN 0-8186-7213-7. (cited on p. 10)

[32] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998. ISSN 0018-9340. (cited on p. 10)

[33] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991. (cited on p. 6)

[34] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977. (cited on p. 7)

[35] R. Dijkman, M. Dumas, B.F. van Dongen, R. Krik, and J. Mendling. Similarity of Business Process Models: Metrics and evaluation. *Information Systems*, 36(2):498–516, 2011. ISSN 0306-4379. (cited on pp. 86, 87, and 200)

[36] G. Dong, J. Han, J.M.W. Lam, J. Pei, and K. Wang. Mining Multi-Dimensional Constrained Gradients in Data Cubes. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 321–330, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4. (cited on p. 8)

[37] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2001. (cited on pp. 6 and 7)

[38] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. SpringerVerlag, 2003. ISBN 3540401849. (cited on pp. 6 and 10)

[39] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J.D. Ullman. Computing Iceberg Queries Efficiently. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 299–310, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1-55860-566-5. (cited on p. 8)

[40] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54, 1996. (cited on pp. 141 and 201)

[41] D. Fisher. Improving Inference through Conceptual Clustering. In *Proceedings of the sixth National conference on Artificial intelligence - Volume 2*, AAAI'87, pages 461–465. AAAI Press, 1987. ISBN 0-934613-42-7. (cited on p. 7)

[42] K. Gerke, J. Cardoso, and A. Claus. Measuring the Compliance of Processes with Reference Models. In *Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part I*, OTM '09, pages 76–93, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-05147-0. (cited on p. 87)

[43] D. Greefhorst and E. Proper. *Architecture Principles: The Cornerstones of Enterprise Architecture*. The Enterprise Engineering Series. Springer, 2011. ISBN 9783642202780. (cited on p. 22)

[44] P.W.P.J. Grefen. Business Information System Architecture. Reader, January 2012. Version Spring 2012. (cited on pp. 21, 22, 24, and 27)

[45] D. Grigori, J.C. Corrales, M. Bouzeghoub, and A. Gater. Ranking BPEL Processes for Service Discovery. *IEEE Transactions on Services Computing*, 3(3):178–192, July 2010. ISSN 1939-1374. (cited on p. 86)

[46] C.W. Günther. *Process Mining in Flexible Environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2009. (cited on pp. 9, 10, 11, and 41)

[47] C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining: Adaptive Process Simplification Based on Multi-perspective Metrics. In *Proceedings of the 5th International Conference on Business Process Management*, BPM'07, pages 328–343, Berlin, Heidelberg, 2007. Springer-Verlag. (cited on pp. 9, 10, 11, 133, and 134)

[48] J. Han. OLAP Mining: An Integration of OLAP with Data Mining. In *In Proceedings of the 7th IFIP 2.6 Working Conference on Database Semantics (DS-7)*, pages 1–9, 1997. (cited on p. 8)

[49] J. Han. Towards on-line analytical mining in large databases. *SIGMOD Record*, 27(1): 97–107, March 1998. ISSN 0163-5808. (cited on p. 8)

[50] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. Elsevier, 2006. ISBN 9781558609013. (cited on pp. 4, 6, 7, 8, 96, 98, 110, 141, 201, and 204)

[51] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM. ISBN 1-58113-217-4. (cited on p. 5)

[52] J. Han, J. Pei, G. Dong, and K. Wang. Efficient computation of Iceberg cubes with complex measures. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 1–12, New York, NY, USA, 2001. ACM. ISBN 1-58113-332-4. (cited on p. 8)

[53] D. Hawking. Challenges in enterprise search. In *Proceedings of the 15th Australasian Database Conference - Volume 27*, ADC '04, pages 15–24, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc. (cited on pp. 30 and 201)

[54] D. Heckerman. Advances in knowledge discovery and data mining. chapter Bayesian Networks for Knowledge Discovery, pages 273–305. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996. ISBN 0-262-56097-6. (cited on p. 6)

[55] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for Association Rule Mining: a General Survey and Comparison. *SIGKDD Explorations Newsletter*, 2(1):58–64, June 2000. ISSN 1931-0145. (cited on p. 6)

[56] K. Huang, Z. Zhou, Y. Han, G. Li, and J. Wang. An Algorithm for Calculating Process Similarity to Cluster Open-Source Process Designs. In H. Jin, Y. Pan, N. Xiao, and J. Sun, editors, *Grid and Cooperative Computing - GCC 2004 Workshops*, volume 3252 of *Lecture Notes in Computer Science*, pages 107–114. Springer Berlin / Heidelberg, 2004. ISBN 978-3-540-23578-1. (cited on p. 86)

[57] S.C. Hui and G. Jha. Data Mining for Customer Service Support. *Information and Management*, 38(1):1–13, 2000. ISSN 0378-7206. (cited on p. 8)

[58] T. Imieliński, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing Association Rules. *Data Mining and Knowledge Discovery*, 6(3):219–257, July 2002. ISSN 1384-5810. (cited on p. 8)

[59] W.H. Inmon. *Building the Data Warehouse,3rd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 3rd edition, 2002. ISBN 0471081302. (cited on p. 26)

[60] W.H. Inmon, C. Imhoff, and R. Sousa. *Corporate Information Factory*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2000. ISBN 0471399612. (cited on p. 27)

[61] R.A. Ittoo, L. Maruster, J.C. Wortmann, and G. Bouma. Textractor: A Framework for Extracting Relevant Domain Concepts from Irregular Corporate Textual Datasets. In W. Abramowicz and R. Tolksdorf, editors, *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 71–82. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12813-4. (cited on p. 5)

[62] S. Jablonski and M. Goetz. Perspective oriented business process visualization. In *Proceedings of the 2007 International Conference on Business Process Management*, BPM'07, pages 144–155, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-78237-0, 978-3-540-78237-7. (cited on p. 16)

[63] R.A. Johnson and D.W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 5th edition, 2002. ISBN 0-130-92553-5. (cited on p. 6)

[64] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley and Sons, New York, 1990. (cited on p. 7)

[65] R. Kimball, L. Reeves, W. Thornthwaite, M. Ross, and W. Thornwaite. *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1998. ISBN 0471255475. (cited on pp. 12, 28, and 96)

[66] E. Kindler, V. Rubin, and W. Schäfer. Process Mining and Petri Net Synthesis. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 105–116. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-38444-1. (cited on p. 10)

[67] A. Koca, M. Funk, E. Karapanos, A. Rozinat, W.M.P. van der Aalst, H. Corporaal, J.B.O.S. Martens, P.H.A. van der Putten, A.J.M.M. Weijters, and A.C. Brombacher. Soft Reliability: An Interdisciplinary Approach with a UserSystem Focus. *Quality and Reliability Engineering International*, 25(1):3–20, 2009. ISSN 1099-1638. (cited on p. 9)

[68] S. Kotsiantis and D. Kanellopoulos. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58, 2006. (cited on p. 151)

[69] S.B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceedings of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, 2007. IOS Press. ISBN 978-1-58603-780-2. (cited on p. 7)

[70] R.W.M. Kuijpers. Merging Field Feedback Data for Product Design Improvement: Enterprise Search Tools versus Ad Hoc Solutions. Master's thesis, Eindhoven University of Technology, Eindhoven, 2011. (cited on p. 34)

[71] C.C. Lee and C. Hu. Analyzing Hotel Customers' E-Complaints from an Internet Complaint Forum. *Journal of Travel and Tourism Marketing*, 17(2-3):167–181, 2004. (cited on p. 5)

[72] X. Li and J. Han. Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 447–458. VLDB Endowment, 2007. ISBN 978-1-59593-649-3. (cited on p. 8)

[73] X. Li, J. Han, and H. Gonzalez. High-dimensional OLAP: A minimal cubing approach. In *Proceedings of the 30th International Conference on Very Large Data Bases*, VLDB '04, pages 528–539, 2004. (cited on pp. 8, 111, and 201)

[74] S.L. Liao. Knowledge Management Technologies and Applications: Literature Review from 1995 to 2002. *Expert Systems with Applications*, 25(2):155–164, 2003. ISSN 0957-4174. (cited on p. 4)

[75] H. Liu, F. Hussain, C.L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 6(4):393–423, October 2002. ISSN 1384-5810. (cited on p. 105)

[76] M. Liu, E. Rundensteiner, K. Greenfield, C. Gupta, S. Wang, I. Ari, and A. Mehta. E-Cube: Multi-dimensional event sequence processing using concept and pattern hierarchies. *Data Engineering, International Conference on*, 0:1097–1100, 2010. (cited on p. 8)

[77] R. Lorenz, R. Bergenthum, J. Desel, and S. Mauser. Synthesis of Petri Nets from Finite Partial Languages. In *Proceedings of the Seventh International Conference on Application of Concurrency to System Design*, ACSD '07, pages 157–166, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-2902-X. (cited on p. 10)

[78] R. Lorenz, S. Mauser, and G. Juhás. How to Synthesize Nets from Languages: a Survey. In *Proceedings of the 39th Conference on Winter Simulation*, WSC '07, pages 637–647, Piscataway, NJ, USA, 2007. IEEE Press. ISBN 1-4244-1306-0. (cited on p. 10)

[79] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif. Quality characteristics for software architecture. *Journal of Object Technology*, 2(2):133–150, 2003. (cited on p. 169)

[80] L. Ma. How to Evaluate the Performance of Process Discovery Algorithms: A Benchmark Experiment to Assess the Performance of Flexible Heuristics Miner. Master's thesis, Eindhoven University of Technology, Eindhoven, 2012. (cited on p. 136)

[81] J.B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967. (cited on p. 7)

[82] E. Maeland. On the Comparison of Interpolation Methods. *IEEE Transactions on Medical Imaging*, 7(3):213 –217, September 1988. ISSN 0278-0062. (cited on p. 104)

[83] S. Mansmann, T. Neumuth, and M.H. Scholl. Multidimensional data modeling for business process analysis. In *Proceedings of the 26th International Conference on Conceptual Modeling*, ER'07, pages 23–38, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3-540-75562-4, 978-3-540-75562-3. (cited on p. 16)

[84] H. Min, H. Min, and A. Emam. A Data Mining Approach to Developing the Profiles of Hotel Customers. *International Journal of Contemporary Hospitality Management*, 14 (6):274–285, 2002. (cited on p. 5)

[85] M. Minor, A. Tartakovski, and R. Bergmann. Representation and Structure-Based Similarity Assessment for Agile Workflows. In R. Weber and M. Richter, editors, *Case-Based Reasoning Research and Development*, volume 4626 of *Lecture Notes in Computer Science*, pages 224–238. Springer Berlin / Heidelberg, 2007. ISBN 978-3-540-74138-1. (cited on p. 86)

[86] G. Monakova and F. Leymann. Workflow ART. In *Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'10, pages 376–393, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16933-3, 978-3-642-16933-5. (cited on p. 16)

[87] E.W.T. Ngai, L. Xiu, and D.C.K. Chau. Application of Data Mining Techniques in Customer Relationship Management: A Literature Review and Classification. *Expert Systems with Applications*, 36(2, Part 2):2592–2602, 2009. ISSN 0957-4174. (cited on p. 4)

[88] OMG. Business Process Model And Notation (BPMN) (Version 2.0). Technical Report OMG Document Number: formal/2011-01-03, Object Management Group, http://www.omg.org/spec/BPMN/2.0/, January 2011. (cited on p. 9)

[89] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986. ISSN 0885-6125. (cited on pp. 6 and 152)

[90] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0. (cited on pp. 6 and 152)

[91] W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1 edition, 1998. ISBN 3540653066. (cited on p. 9)

[92] J.T.S. Ribeiro and A.J.M.M. Weijters. Event cube: Another Perspective on Business Processes. In *Proceedings of the 2011th Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'11, pages 274–283, Berlin, Heidelberg, 2011. Springer-Verlag. (cited on p. 34)

[93] A. Rozinat. *Process Mining: Conformance and Extension*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2010. (cited on pp. 11 and 43)

[94] A. Rozinat and W.M.P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In *Proceedings of the 3rd International Conference on Business Process Management*, BPM'05, pages 163–176, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-32595-6. (cited on p. 11)

[95] A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In *Proceedings of the 4th International Conference on Business Process Management*, BPM'06, pages 420–425, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-38901-6. (cited on pp. 10 and 11)

[96] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, March 2008. ISSN 0306-4379. (cited on p. 11)

[97] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X. (cited on p. 6)

[98] J. Sander, M. Ester, H.P. Kriegel, and X. Xu. Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, June 1998. ISSN 1384-5810. (cited on p. 7)

[99] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-Driven Exploration of OLAP Data Cubes. In H.J. Schek, G. Alonso, F. Saltor, and I. Ramos, editors, *Advances in Database Technology EDBT'98*, volume 1377 of *Lecture Notes in Computer Science*, pages 168–182. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64264-0. (cited on p. 203)

[100] B. Schnhage, A. van Ballegooij, and A. Elins. 3D gadgets for business process visualization - a case study. In *Proceedings of the 5th Symposium on Virtual Reality Modeling Language (Web3D-VRML), United States*, 2000. (cited on p. 16)

[101] C.E. Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27, 1948. (cited on p. 151)

[102] A. Sidi. *Practical Extrapolation Methods: Theory and Applications*. Cambridge University Press, New York, NY, USA, 1st edition, 2002. ISBN 0521661595, 9780521661591. (cited on p. 104)

[103] M. Song and W.M.P. van der Aalst. Supporting Process Mining by Showing Events at a Glance. *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145, 2007. (cited on p. 9)

[104] M. Song and W.M.P. van der Aalst. Towards Comprehensive Support for Organizational Mining. *Decision Support Systems*, 46(1):300–317, December 2008. ISSN 0167-9236. (cited on p. 9)

[105] P. Tan, V. Kumar, and J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, June 2004. ISSN 0306-4379. (cited on p. 147)

[106] A. Tiwari, C.J. Turner, and B. Majeed. A Review of Business Process Mining: State-of-the-Art and Future Trends. *Business Process Management Journal*, 14(1):5–22, 2008. (cited on p. 16)

[107] K.T. Ulrich and S.D. Eppinger. *Product Design and Development*. McGraw-Hill/Irwin series in marketing. McGraw-Hill/Irwin, 2003. ISBN 9780071232739. (cited on p. 4)

[108] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. (cited on p. 10)

[109] W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999. ISSN 0950-5849. (cited on p. 9)

[110] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, Berlin, 2011. ISBN 978-3-642-19344-6. (cited on pp. 8, 9, 16, and 43)

[111] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47:237–267, November 2003. ISSN 0169-023X. (cited on p. 9)

[112] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, September 2004. ISSN 1041-4347. (cited on pp. 9 and 10)

[113] W.M.P. van der Aalst, A.K. A. de Medeiros, and A.J.M.M. Weijters. Genetic Process Mining. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 48–69. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26301-2. (cited on p. 10)

[114] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work*, 14(6):549–593, December 2005. ISSN 0925-9724. (cited on p. 9)

[115] W.M.P. van der Aalst, M. Dumas, C. Ouyang, A. Rozinat, and H.M.W. Verbeek. Conformance checking of service behavior. *ACM Transactions on Internet Technology*, 8(3): 13:1–13:30, May 2008. ISSN 1533-5399. (cited on p. 11)

[116] W.M.P. van der Aalst, V. Rubin, H.M.W. Verbeek, B.F. van Dongen, E. Kindler, and C.W. Günther. Process mining: A two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 9:87–111, 2010. ISSN 1619-1366. (cited on p. 10)

[117] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Causal nets: A modeling language tailored towards process discovery. In *Proceedings of the 22nd International Conference on Concurrency Theory*, CONCUR'11, pages 28–42, Berlin, Heidelberg, 2011. Springer-Verlag. (cited on pp. 43 and 200)

[118] W.M.P. van der Aalst, A. Adriansyah, A.K. A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, J.C. Bose, P. Brand, R. Brandtjen, J.C.A.M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani,

M. Leoni, P. Delias, B.F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F. Geffen, S. Goel, C.W. Günther, A. Guzzo, P. Harmon, A. Hofstede, J. Hoogland, J.E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. Rosa, F. Maggi, D. Malerba, R.S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R. Motahari-Nezhad, M. Muehlen, J. Munoz-Gama, L. Pontieri, J.T.S. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, H.M.W. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, A.J.M.M. Weijters, L. Wen, M. Westergaard, and M. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28107-5. (cited on p. 16)

[119] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery Using Integer Linear Programming. In *Proceedings of the 29th International Conference on Applications and Theory of Petri Nets*, PETRI NETS '08, pages 368–387, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-68745-0. (cited on p. 10)

[120] B.F. van Dongen and W.M.P. van der Aalst. A meta model for process mining data. In *Proceedings of the CAiSE 2005 Workshops*, EMOI-INTEROP Workshop. (cited on p. 41)

[121] B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In *International Conference on Conceptual Modeling (ER 2004), volume 3288 of Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, 2004. (cited on p. 10)

[122] B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets. *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB)*, pages 35–38, 2005. (cited on p. 10)

[123] B.F. van Dongen, A.K. A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *ICATPN*, pages 444–454, 2005. (cited on pp. 84, 94, 108, 118, 139, 168, and 174)

[124] B.F. van Dongen, A.K. A. de Medeiros, and L. Wen. Transactions on petri nets and other models of concurrency ii. chapter Process Mining: Overview and Outlook of Petri Net Discovery Algorithms, pages 225–242. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-00898-6. (cited on p. 10)

[125] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The Process Mining Toolkit. In *Demo at the 8th International Conference on Business Process Management*. (cited on pp. 84, 94, 108, 139, 168, and 174)

[126] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In P. Soffer, E. Proper, W.M.P. van der Aalst, J. Mylopoulos, M. Rosemann, M.J. Shaw, and C. Szyperski, editors, *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-17722-4. (cited on p. 41)

[127] W. Wang, J. Yang, and R.R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 186–195, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc. ISBN 1-55860-470-7. (cited on p. 7)

[128] H.J. Watson and T. Ariyachandra. Data warehouse architectures: Factors in the selection decision and the success of the architectures. Technical report, Terry College of Business, University of Georgia, July 2005. (cited on p. 27)

[129] M. Weidlich, J. Mendling, and M. Weske. Efficient Consistency Measurement Based on Behavioral Profiles of Process Models. *IEEE Transactions on Software Engineering*, 37 (3):410–429, May 2011. ISSN 0098-5589. (cited on p. 86)

[130] A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible Heuristics Miner (FHM). In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, Paris, France*. IEEE, 2011. (cited on pp. 9, 34, 43, and 133)

[131] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2): 151–162, April 2003. ISSN 1069-2509. (cited on pp. 10, 11, 61, 83, and 201)

[132] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K. A. de Medeiros. Process Mining with the HeuristicsMiner Algorithm. Technical Report 166, Eindhoven University of Technology, 2006. (cited on pp. 10, 11, 61, 83, 136, and 201)

[133] L. Wen, J. Wang, and J. Sun. Detecting Implicit Dependencies Between Tasks from Event Logs. In X. Zhou, J. Li, H. Shen, M. Kitsuregawa, and Y. Zhang, editors, *Frontiers of WWW Research and Development - APWeb 2006*, volume 3841 of *Lecture Notes in Computer Science*, pages 591–603. Springer Berlin / Heidelberg, 2006. ISBN 978-3-540-31142-3. (cited on p. 10)

[134] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Data Management Systems. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2005. ISBN 0120884070. (cited on p. 5)

[135] D. Xin, J. Han, X. Li, and B.W. Wah. Star-cubing: Computing iceberg cubes by top-down and bottom-up integration. In *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, VLDB '03, pages 476–487. VLDB Endowment, 2003. ISBN 0-12-722442-4. (cited on p. 8)

[136] D. Xin, Z. Shao, J. Han, and H. Liu. C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE '06, pages 4–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2570-9. (cited on p. 8)

[137] D. Xin, J. Han, X. Li, Z. Shao, and B.W. Wah. Computing Iceberg Cubes by Top-Down and Bottom-Up Integration: The StarCubing Approach. *Knowledge and Data Engineering, IEEE Transactions on*, 19(1):111–126, jan. 2007. ISSN 1041-4347. (cited on p. 8)

[138] R. Xu and II Wunsch. Survey of Clustering Algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005. ISSN 1045-9227. (cited on p. 7)

[139] Z. Yan. *Business Process Model Repositories: Efficient Process Retrieval*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2012. (cited on p. 85)

[140] Z. Yan, R. Dijkman, and P.W.P.J. Grefen. Fast business process similarity search with feature-based similarity estimation. In *Proceedings of the 2010 International Conference on On the Move to Meaningful Internet Systems - Volume Part I*, OTM'10, pages 60–77, Berlin, Heidelberg, 2010. Springer-Verlag. (cited on p. 85)

[141] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965. (cited on p. 6)

[142] M.J. Zaki. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May 2000. ISSN 1041-4347. (cited on p. 5)

[143] C. Zhang and S. Zhang. *Association Rule Mining: Models and Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3-540-43533-6. (cited on p. 6)

[144] Y. Zhao, P.M. Deshpande, and J.F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, SIGMOD '97, pages 159–170, New York, NY, USA, 1997. ACM. ISBN 0-89791-911-4. (cited on p. 8)

[145] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), July 2006. ISSN 0360-0300. (cited on pp. 100 and 201)

Summary

Multidimensional Process Discovery

Typically represented in event logs, business process data describe the execution of process events over time. Business process intelligence (BPI) techniques such as process mining can be applied to get strategic insight into business processes. Process *discovery*, *conformance* checking and *enhancement* are possible applications for knowledge discovery on process data. By applying process mining techniques on data that describe the behavior of process instances, it is possible to discover and analyze the business as it is being executed. However, so far, these techniques are typically designed to focus on specific process dimensions, omitting information potentially relevant for the analysis comprehension.

Specially designed to support dynamic hypothesis-driven exploration of data, online analytic processing (OLAP) systems are commonly used as reporting tools in almost every application for business intelligence. Exploiting the data by combining different dimensions with some measures of interest, it is possible to adjust on-the-fly the analysis' level of abstraction in an interactive way. Pivot tables are a good example of reporting tools that support hypothesis-driven analysis. One typical application of these tables is the sales analysis where the *total amount of sales* in a company can be characterized dynamically by multiple dimensions (e.g., *product*, *customer*, and *time*).

This thesis presents our research into the extension of process mining techniques with OLAP functionalities. Making use of OLAP concepts, a multidimensional data structure is designed to hold the different dimensions of business processes in such a way that process discovery and analysis are facilitated. Extending the traditional OLAP data cubes, these structures can improve the process analysis by providing immediate results under different levels of abstraction. For example, the process behavior described in a process model can be constrained by *activity*, *resource*, or *time*, or by any combination of these dimensions.

The dynamic integration of business process dimensions into process mining can bring process analysis to another level. Considering the event log's attributes as dimensions, process analysis can be performed in such a way that events and workflow can be constrained by specific process information. Traditionally, only a single dimension (or a static set of dimensions) is considered. For example, the *activity* is typically the only considered dimension in a process model. In a multidimensional approach, process analysis can be dynamically constrained by the dimensions the analyst considers relevant. The elements of a process model (i.e., nodes describing process events and

arcs representing the workflow) can be constrained by any set of dimensions. Eventually, these elements can also be annotated with measures of interest. Both dimensions and measures of interest are described in – or derived from – the process data.

The work presented in this thesis is supported by concrete implementations as plug-ins in the ProM framework[1], and has been applied to real-life case studies. Besides the multidimensional data structure described above, a process discovery technique based on the Heuristics Miner algorithm is introduced to demonstrate the applicability of process mining techniques in a multidimensional context. By building a data cube of process events, it is possible to construct on-the-fly – multidimensional – process models according to specific process perspectives and levels of abstraction. Furthermore, by applying data mining functionalities, it is possible to extract non-trivial patterns from multidimensional process models. A characterization of these patterns as well as a number of techniques for process pattern extraction are presented in this thesis. A qualitative and quantitative evaluation is conducted to demonstrate the feasibility of the multidimensional approach as well as its added value to the stakeholders.

[1] `www.processmining.org`

Samenvatting

Multidimensional Process Discovery

Bij het uitvoeren van bedrijfsprocessen kunnen we, in een zogenaamde *event log*, registreren op welk moment welke activiteit wordt uitgevoerd. Business Process Intelligence (BPI) technieken zoals *process mining* kunnen gebruikt worden om deze event logs te analyseren, om zo een beter inzicht te verkrijgen in het betreffende bedrijfsproces. Process *discovery* (proces reconstructie), *conformance checking* (het bepalen van de mate van overeenstemming tussen event log en proces model) en *enhancement* (uitbreiding of verbetering van het proces model) zijn mogelijke process mining toepassingen. Door analyseren van event logs met behulp van process mining technieken is het mogelijk een beter inzicht te verkrijgen in wat er werkelijk gebeurt tijdens het uitvoeren van een bedrijfsproces. Echter, de tot nu toe ontwikkelde process mining technieken richten zich veelal op een of meerdere specifieke proces dimensies, waardoor andere potentieel bruikbare proces dimensies worden verwaarloosd.

Traditioneel worden bedrijfsgegevens opgeslagen in database systemen. Online analytic processing (OLAP) systemen zijn special ontwikkeld voor het flexibel en hypothese-gestuurd analyseren van dit soort bedrijfsgegevens. Gebruikmakend van relevante dimensies (omzet, tijdsintervallen, regio's, enz.) is het mogelijk interactief de bedrijfsgegevens vanuit verschillende invalshoeken te analyseren. Zogenaamde Pivot tabellen zijn een goed voorbeeld van een hypothese-gestuurde analyse techniek. Deze Pivot tabellen kunnen gebruikt worden om de verkoopcijfers van een bedrijf te analyseren waarbij gebruik wordt gemaakt van meerdere dimensies zoals product, klant en tijd.

In dit proefschrift presenteren we ons onderzoek om process mining technieken uit te breiden met de beschreven OLAP functionaliteit. Gebruikmakend van OLAP concepten, is een data structuur ontworpen die multidimensional process discovery en analyse mogelijk maakt. Gebruikmakend van deze nieuwe multidimensionale data structuur is het mogelijk zeer flexibel proces analyses uit te voeren waarbij gebruik wordt gemaakt van relevante proces aspecten en verschillende niveaus van abstractie. Zo is het mogelijk proces modellen te genereren waarbij de focus ligt op de activiteiten, de uitvoerders van activiteiten, of tijdsaspecten. Ook is het mogelijk beschikbare dimensies (proces aspecten) te combineren.

De dynamische integratie van de verschillende dimensies van bedrijfsprocessen maakt een andere manier van process mining mogelijk. Door attributen die zijn opgenomen in een event log te beschouwen als dimensies, is het mogelijk deze te

gebruiken bij het analyseren van een event log. Traditioneel, wordt tijdens process mining enkel gebruik gemaakt van één enkele dimensie of een vaste combinatie van dimensies. Bijvoorbeeld, de activiteit, is normaal de enige dimensie die wordt gebruikt bij mining van een proces model. In de hier ontwikkelde multidimensionale aanpak kan de analist dynamisch de dimensies kiezen die hij in de gegeven situatie relevant acht. De verzameling van elementen van een proces model (d.w.z. knopen die activiteiten aangeven en verbindingen die overgangen representeren) kan beïnvloed worden met behulp van iedere mogelijke combinatie van dimensies. Bovendien kunnen elementen van het proces model voorzien worden van relevante extra informatie zoals frequenties, gemiddelde tijden, enz. Deze extra informatie wordt afgeleid uit de data in de event log.

Het werk gepresenteerd in dit proefschrift wordt ondersteund door de implementatie van verschillende plug-ins in het ProM framework[2]. Deze plug-ins zijn toegepast in enkele case studies op data afkomstig van real-life bedrijfsprocessen. Naast de hiervoor beschreven multidimensionale data structuur (de zogenaamde *data cube*) wordt in het proefschrift ook een process discovery techniek geïntroduceerd die gebaseerd is op het Heuristics Miner algoritme. Door dit process discovery algoritme toe te passen op een data cube met process events is het mogelijk on-the-fly multidimensionale proces modellen te genereren vanuit een gespecificeerd proces perspectief en van het gewenste abstractie niveau. Door het toevoegen van traditionele data mining functionaliteit is het mogelijk niet-triviale patronen in multidimensionale proces modellen te achterhalen. Enkele technieken voor het achterhalen van patronen in proces modellen worden in dit proefschrift gepresenteerd en aangevuld met een typologie van deze patronen. Tenslotte volgt een kwalitatieve en kwantitatieve evaluatie van de voorgestelde aanpak. Deze evaluatie laat zien dat multidimensional process mining realistisch is, en laat ook zien dat deze techniek voor proces eigenaren bruikbare inzichten oplevert.

[2] `www.processmining.org`

Acknowledgements

Many people have contributed to this thesis. Here, I would like to take the opportunity to express my gratitude to them.

First of all, I would like to thank my supervisors Ton Weijters and Paul Grefen for giving me the opportunity to pursue this PhD and for their guidance throughout the last four years. It was really a pleasure to work with you. Secondly, I want to thank the people involved in the DataFusion project with whom I had the privilege to collaborate with. From the academic side, they are Lu Yuan, Ashwin Ittoo, Renate de Bruin, Eva Hopma, Matthijs Zwinderman, Hans Wortmann, Aarnout Brombacher, and Laura Maruster. From the industry side, they are Leif Sørensen, Susanne Korsch, Karl Kurz, Ludwig Nooyens, Guillaume Stollman, Frank Spronck, Edwin Noordman, Gert van Lieshout, and Irina Sepkhanova. From AgentschapNL, they are Michiel de Boer and Joop Postema. I would also like to thank Roel Kuijpers and Lulu Ma for their contribution as MSc students.

Further, I would like to thank the people with whom I have been collaborating with at the IS group at the faculty of Industrial Engineering & Innovation Sciences. In particular, I would like to thank Zhiqiang Yan, Heidi Romero, Jana Šamalíková, Marco Comuzzi, Vassil Stoitsev, Egon Lüftenegger, Ronny Mans, Shaya Pourmirza, Nan Shan, Hui Yan, Rob Vanwersch, Rik Eshuis, and Dániel Kelemen for their support and companionship. Thanks also to the secretaries of the group Annemarie van der Aa and Ada Rijnberg-Wielaard for their help and assistance. I would also like to thank the people from the process mining community for their discussions and feedback. In particular, I want to thank Arya Adriansyah, Wil van der Aalst, Boudewijn van Dongen, Eric Verbeek, Anne Rozinat, and Christian Günther.

Last but not least, I would like to express my thanks and gratitude to my family for their support and guidance throughout my life. Also, I would like to express my sincere gratitude to Luísa Benta and her family for their support over many years. Finally, I want to thank my friends Joana Laranja, Angela Félix, Nuno Ribeiro, José Fontão, José Moreira, and Jorge Santos for always being there.

Curriculum Vitae

Joel Ribeiro was born on 5 August 1980 in Vila de Cucujães, Portugal. From 1995 to 1999 he attended secondary school at the Escola Secundária Dr. Serafim Leite in São João da Madeira.

After finishing secondary school, Joel studied Informatics and Systems Engineering at University of Minho in Braga, Portugal from 2000 to 2006. He obtained his Licentiate degree in September 2006 after completing an internship about detection and analysis of churn in a telecom company, under supervision of prof. dr. O.M.O. Belo. The following two years, he followed a Master's program in Data Systems and Analytic Processing at University of Minho, receiving his Master of Science degree in November 2008. The title of his Master's thesis, supervised by prof. dr. O.M.O. Belo, is "Multidimensional Top-k Gradients".

In December 2008, Joel became a PhD candidate in the department of Industrial Engineering and Innovation Sciences at Eindhoven University of Technology in the Netherlands. The focus of his doctoral studies, supervised by prof. dr. ir. P.W.P.J. Grefen and dr. A.J.M.M. Weijters, was on the multidimensional discovery and analysis of business processes. He completed his doctoral studies in 2013 with a thesis titled "Multidimensional Process Discovery".

Since 2006, Joel has been involved in a number of industry projects such as FRATELO (fraud detection on telecommunication systems), SFonTel (graph mining in telecommunication data), and DataFusion[3] (merging of incoherent field feedback data into prioritized design information). Joel can be reached at joel.ribeiro@mail.com.

[3] `www.iopdatafusion.org`