TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Process mining based on object-centric behavioral constraint (OCBC) models

Citation for published version (APA):
Li, G. (2019). *Process mining based on object-centric behavioral constraint (OCBC) models*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Mathematics and Computer Science].

Document status and date:
Published: 14/05/2019

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Jul. 2024

Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models

Guangming Li

Copyright © 2019 by Guangming Li. All Rights Reserved.

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Li, Guangming

Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models by Guangming Li.
Eindhoven: Technische Universiteit Eindhoven, 2019. Proefschrift.

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

ISBN: 978-90-386-4752-4

Keywords: Process Mining, Databases, Object-Centric Behavioral Constraints, Model Discovery, Conformance Checking, Performance Analysis

SIKS Dissertation Series No. 2019-16
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Printed by IPSKAMP printing

Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor een commissie aangewezen door het College voor Promoties, in het openbaar te verdedigen op dinsdag 14 mei 2019 om 13:30 uur

door

Guangming Li

geboren te Kaifeng, China

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. Mark G. J. van den Brand
1e promotor:	prof.dr.ir. Wil M. P. van der Aalst
co-promotor:	dr. Renata Medeiros de Carvalho
leden:	prof.dr.ir. Boudewijn F. van Dongen
	dr. George H. L. Fletcher
	prof.dr. Hafedh Mili (Université du Québec à Montréal)
	dr. Marco Montali (Free University of Bozen-Bolzano)
	Prof.dr.ir. Walid Gaaloul (Télécom SudParis)

Het onderzoek of ontwerp dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

To my parents…

Abstract

Nowadays, information systems are widely used in various organizations to support the execution of their business processes. Therefore, the amount of data being stored about process executions is rapidly growing. Today's main innovations are intelligently exploiting the data and turning data into real value. Process mining appears to leverage execution data to analyze the business processes executed on information systems. Existing process mining techniques make a fundamental assumption about processes. Process models and event logs assume the presence of a well-defined case notion. This implies that each event refers to a case and that the model describes the life-cycle of cases. This assumption is consistent with case-centric information systems, e.g., Workflow Management (WFM) systems. However, most information systems one encounters in enterprises nowadays are artifact-centric (or data-centric), e.g., Enterprise Resource Planning (ERP) systems, which do not assume a case notion in their business processes. Such differences lead to problems when applying existing process mining techniques, e.g., it is difficult to identify the case notion for the whole process, and the many-to-many relations cannot be well described. In order to solve these problems, we propose a series of process mining techniques in this thesis, such as a novel log format named eXtensible Object-Centric (XOC) to organize the data generated by artifact-centric information systems and a novel modeling language which combines data/object modeling languages (ER, UML, or ORM) and declarative languages (Declare). Besides, we propose approaches to automatically discover OCBC models from XOC logs, to check conformance and to analyze the performance. In summary, this thesis proposes new process mining techniques to analyze the data from artifact-centric information systems

and the results derived by these techniques provide insights to the process owners. For instance, the discovered process models present behavioral and data perspectives to reveal the real execution of business processes. By checking conformance, the deviating behaviors are detected. Using performance analysis techniques, the bottlenecks are shown and highlighted in the models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, information systems are widely used in various organizations to support the execution of their business processes. Therefore, the amount of data being stored about process executions is rapidly growing. The term "Big Data", which is often used to refer to the incredible growth of data, has become a hot topic in industry and academia. Today's main innovations are intelligently exploiting the data and turning data into real value, rather than easily collecting more data. In other words, data should be analyzed to discover insights for improving existing products, processes and services.

Process mining appears to leverage execution data to analyze the business processes executed on information systems, as shown in Figure 1.1. It provides methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes in enterprises. For instance, it can discover models to show real behavior, check conformance to diagnose deviations and analyze performance to detect bottlenecks.

Conventional process modeling and process mining techniques assume a single case notion. Each case corresponds to an instance of the process. As a result, each recorded event refers to precisely one case. Consequently, cases can be viewed as a sequence of events (traces). Moreover, a process model describes the process from the viewpoint of this case notion, i.e., it describes the life-cycle of a case in isolation, as shown in Figure 1.2. However, the majority of processes and systems are not that simple. Enterprise Resource Planning (ERP) systems and Customer Relationship Management (CRM) systems are typical information systems one encounters in enterprises nowadays, which are artifact-centric (or

data-centric) and do not assume a case notion in their business processes. Such differences lead to problems when applying existing process mining techniques to these systems. *Therefore, in this thesis, we propose a series of process mining techniques, such as a novel log format and modeling language, to analyze data from artifact-centric information systems*.

This chapter generally shows the story-line of this thesis. Section 1.1 briefly introduces the conventional process mining techniques. Then we clarify the target application fields, i.e., artifact-centric information systems, and discuss their differences from case-centric information systems in Section 1.2. Based on the explained difference, Section 1.3 lists the problems faced by conventional

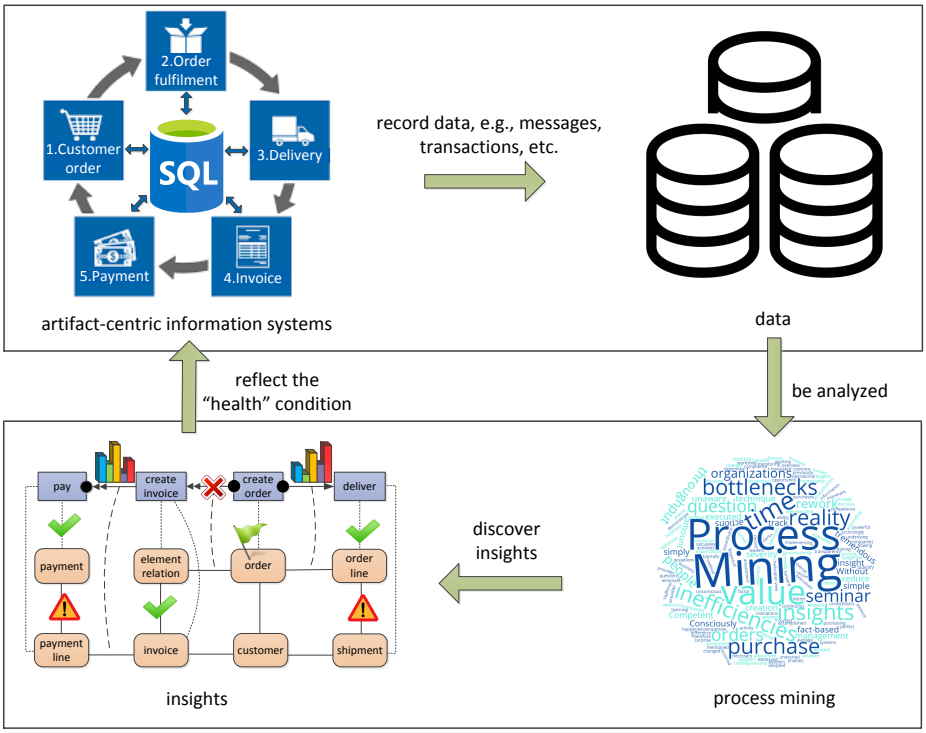


Figure 1.1: Process mining techniques are used to analyze data generated by artifact-centric information systems and discover insights to reflect the "health" condition of systems.

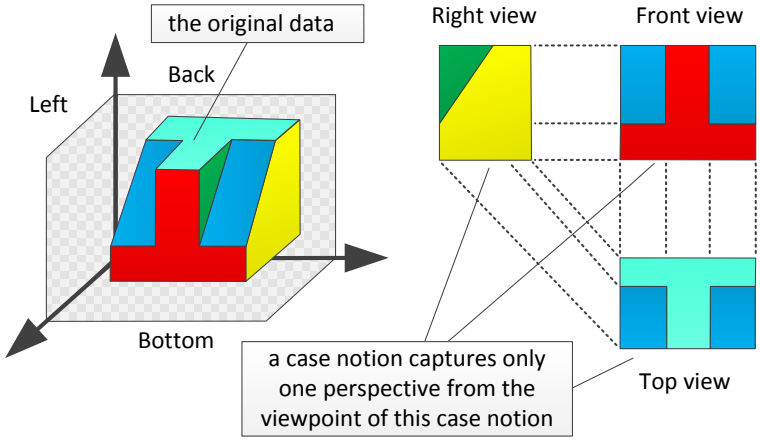


Figure 1.2: Conventional process mining techniques assume a single case notion. They project the 3D data onto a 2D view (i.e., flattening the 3D data) thus capturing only one perspective from the viewpoint of this case notion.

process mining techniques. Section 1.4 provides an overview of the approaches proposed in this thesis and explains how they contribute to solving the existing problems. Section 1.5 presents the structure of the thesis.

1.1 Process Mining

Understanding, analyzing, and ultimately improving business processes based on execution data is a goal of enterprises today. Process mining is a relatively young research discipline that bridges the gap between machine learning and data mining on the one hand and process modeling and analysis on the other hand [136]. The aim of process mining is to automatically provide an accurate view on how the process is executed, by using historical facts as recorded by the information system.

Figure 1.3 shows an overview of the research area covered by process mining. The starting point of process mining is the observed behavior of process executions, stored in so-called *event logs*. Based on event logs, various process mining techniques can be employed to reveal insights. In general, these techniques can be organized into three categories: discovery, conformance and performance

analysis, and enhancement. The first type of process mining techniques is process discovery, which automatically constructs a process model to describe the real behavior using event logs. Next to process discovery, performance and conformance analysis can be implemented to detect deviations and measure performance. Enhancement attempts to combine the results of conformance and performance analysis with the process model, i.e., to project the measured results back onto the model at the right place. The insights derived by process mining techniques, such as discovered (and enhanced) process models, detected deviations and derived performance measurements, reflect the "health" condition of the business processes supported by information systems (indicated by the dotted lines in Figure 1.3). Based on the insights, proper decisions can be made to improve the business processes.

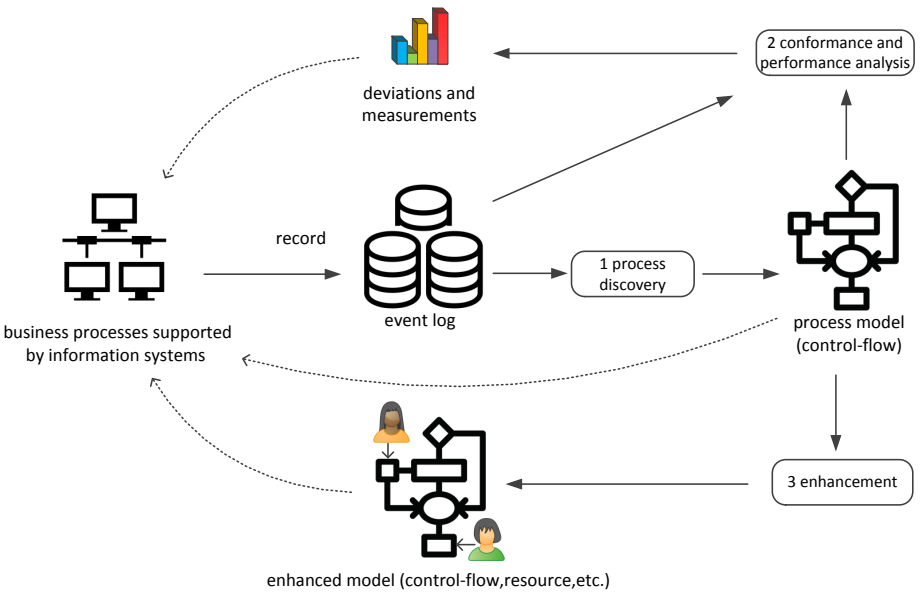


Figure 1.3: Three main types of process mining: process discovery, conformance and performance analysis, and enhancement.

Event logs. Process mining methods typically assume that execution data are stored in event logs. An event log consists of traces, where each trace is a sequence of events and represents one end-to-end execution of a business process. In a trace, each event corresponds to an operation executed in the

process and relates to a particular process step, called an activity. Table 1.1 shows an excerpt of an example event log, containing events generated in the order-to-cash (OTC) business process of an ERP system. Each row in the table represents one event and each column represents an attribute of this event. For instance, the first line represents one "create order" event, referred to by "35654423" (i.e., event id), happening at "2017-08-11 10:33:37" and executed by "Pete". Table 1.1 shows two traces. For instance, the second trace with case id "2" starts with a "create order" event, followed by a "create shipment" event and a "create payment" event.

Case id	Event id	Properties			
		Timestamp	Activity	Resource	...
1	35654423	2017-08-11 10:33:37	create order	Pete	...
	35654424	2017-08-14 11:36:35	create shipment	Mike	...
	35654425	2017-08-15 09:13:27	create invoice	Sue	...
	35654426	2017-08-16 14:15:31	create shipment	Mike	...
	35654427	2017-08-17 17:38:36	create invoice	Sue	...
	35654428	2017-08-21 16:26:13	create payment	Pete	...
2	35654483	2017-08-13 16:28:15	create order	Pete	...
	35654485	2017-08-19 13:22:04	create shipment	Mike	...
	35654488	2017-08-26 14:53:49	create payment	Sara	...
...	...	...	...	...	...

Table 1.1: A segment of an XES log.

Discovery. An ultimate goal in the field of process mining is to automatically discover an accurate and understandable process model based solely on the data recorded in an event log. Process models are used to describe the behavior of processes of an organization for a wide range of objectives such as: communication among stakeholders, process performance analysis and process improvement. Such a process model is typically expressed in a formalism such as a Petri net or a BPMN diagram. Figure 1.4 shows a discovered Petri net based on an event log recording events from the OTC business process of an ERP system.

Conformance and performance analysis. Taking a process model and an event log of the same process as input, conformance checking provides diagnostic information and quantification of discrepancies between the observed behavior in the log and the behavior allowed by the model discovered or manually created

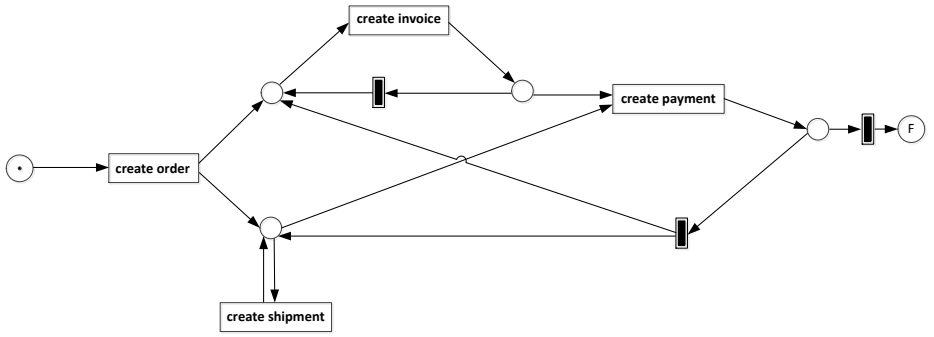


Figure 1.4: A discovered Petri net to describe the OTC business process.

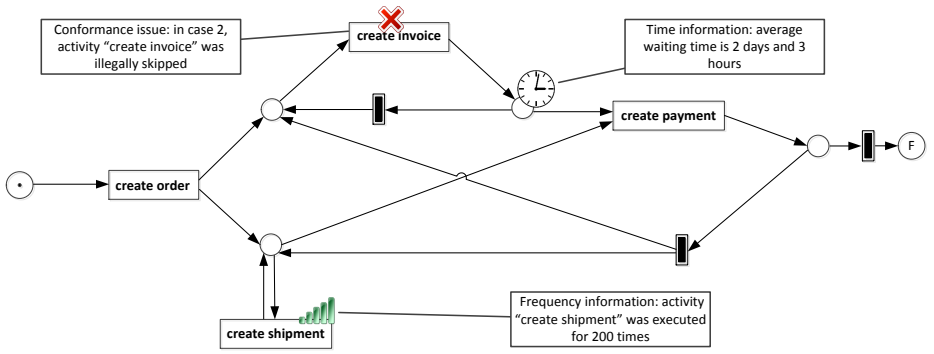


Figure 1.5: Conformance and performance analysis based on the discovered model.

(e.g., a reference model). When the event log and process model do not agree, these discrepancies might indicate undesirable deviations, fraud, inefficiencies, or other issues. For instance, Figure 1.5 shows that activity "create invoice" is violated since there is no corresponding event in case 2 in Table 1.1. Through conformance checking, events in the log are coupled to elements in the model. Based on this, performance analysis measures the performance of a process in terms of times and frequencies. For instance, after replaying the log on the model, durations, service times, waiting times, resource usage, and frequent parts can be derived and evaluated.

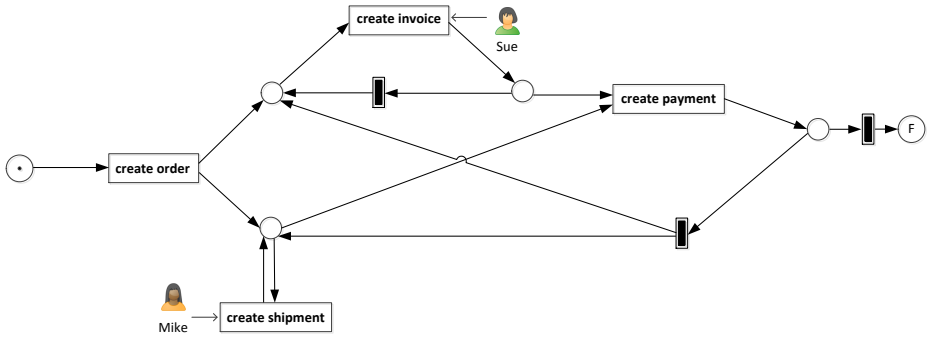


Figure 1.6: Enhancement of the discovered model.

Enhancement. Enhancement techniques take both an event log and a process model as input, to extend or improve the model with information extracted from the event log. There are various enhancements one can perform. One type of enhancement is repair, i.e., modifying the model based on conformance checking results in order to better reflect the real process execution. For example, if two activities are modeled sequentially but in reality can happen in any order, then the model may be corrected to reflect this. Another type of enhancement is extension, i.e., adding new perspectives to the process model or extending the model with information based on the process context, such as frequencies, times, resources, decision rules and performance indicators. For instance, the resource perspective can be added to the model to indicate who executes a particular activity, e.g., Sue executes activity “create invoice”, as shown in Figure 1.6.

1.2 Artifact-Centric Information Systems

Artifact-centric information systems are widely employed in enterprises nowadays, such as Enterprise Resource Planning (ERP) systems and Customer Relationship Management (CRM) systems. Therefore, in this thesis, we choose the artifact-centric information systems as the target application domain for process mining techniques introduced in Section 1.1.

In this section, we first position artifact-centric information systems in the spectrum of process-aware information systems. Then some typical examples of artifact-centric information systems are introduced. Through a comparison

with case-centric information systems, the features of artifact-centric information systems are present.

1.2.1 Positioning Artifact-Centric Information Systems

Business processes are central to the operations of any enterprise [23, 146]. To support business processes, an information system needs to be aware of these processes, which is called *Process-Aware Information System (PAIS)* [40, 133].

Some examples of PAISs are Business Process Management (BPM) systems, WFM (Workflow Management) systems, ERP systems (SAP, Oracle, etc.), CRM systems, rule-based systems, call center software, high-end middleware (WebSphere), etc. What these systems have in common is that there exists a process notion (not just isolated activities) present in the software (e.g., the completion of one activity triggers the start of another activity) and that the information system is aware of the processes it supports (e.g., collecting information about flow times). This is very different from systems such as a e-mail program, database management system (DBMS), text editor, or spreadsheet program. The latter example systems can only be used to execute steps in some business process, but are not "aware" of the processes they are used in [136].

A particular class of PAISs is driven by explicit process models, i.e., the notion of a process model is foundational for these systems. A process model assumes a well-defined case notation for the business process and aims to describes the life-cycle of cases (i.e., process instances). Since these systems assume a case notion, we classify them as case-centric information systems in this thesis. Examples of case-centric information systems are WFM and BPM systems. WFM primarily focuses on the automation of business processes [69, 149], whereas BPM has a broader scope: from process automation and process analysis to process management and the organization of work [39, 161].

The assumption of a single case notion in business processes is too strict for many enterprises, which limits the usage of WFM/BPM systems. In the last decade, ERP systems have derived stronger acceptance and deployment around the world than WFM/BPM systems. According to Aberdeen Group's estimates, the spending in the BPM software sector was $2.26 billion in 2001 [22]. In comparison, ERP systems revenue was $21.5 billion in 2000, according to the research company International Data Corp (IDC) [77]. Besides, as predicted, the dominant position of ERP systems will remain for the next decade.

In addition to ERP systems, there exist some similar information systems, e.g., Customer Relationship Management (CRM) systems. These systems do not assume a case notion for the whole business process. In this thesis, we call these

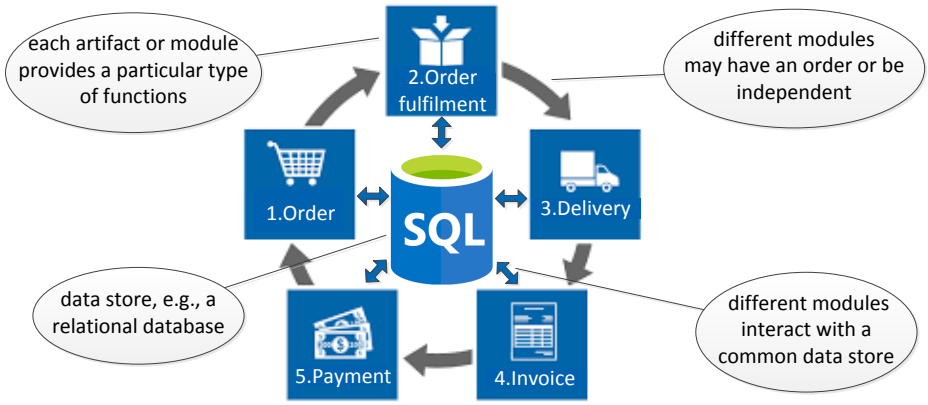


Figure 1.7: The architecture of artifact-centric information systems.

systems artifact-centric information systems, which are another class of PAISs, as shown in Figure 1.7 and defined as follows:

> *a software system that consists of multiple artifacts (or modules), each providing a particular type of functions to manage and execute operational processes involving people, applications, and/or information sources.*

Based on the above discussion, Figure 1.8 presents the position of artifact-centric information systems in the scope of information systems. In general, process mining can be applied to all PAISs. Currently, most use cases for process mining (existing application and future applications) focus on artifact-centric information systems and do not involve case-centric information systems. However, existing process mining techniques make a fundamental assumption of a case notion, i.e., they analyze the data from artifact-centric systems in a case-centric manner. As a result, this mismatch may lead to some problems, that are discussed in Section 1.3.

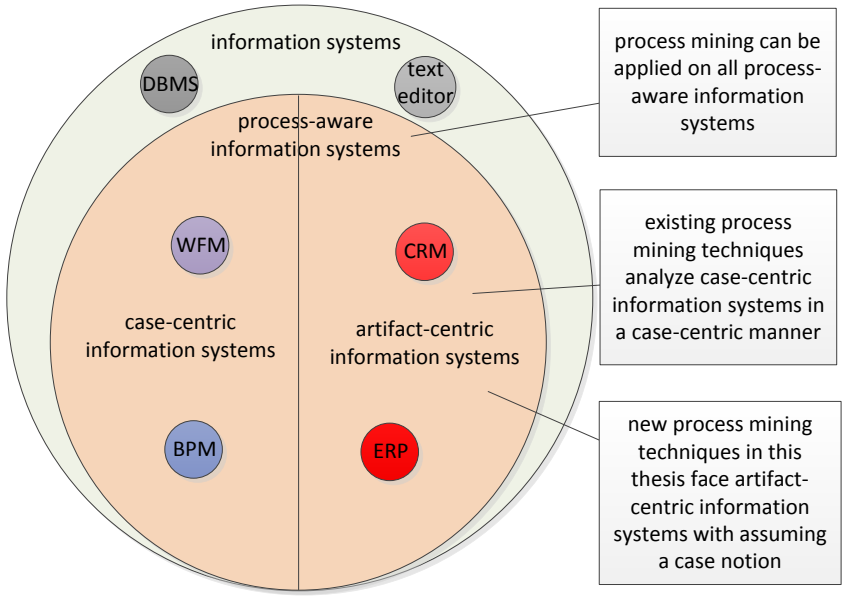


Figure 1.8: Positioning artifact-centric information systems.

1.2.2 Typical Examples of Artifact-Centric Information Systems

Artifact-centric information systems have a big size of market share. Some famous examples are offered as commercial packages from various software vendors such as Microsoft,[1] Oracle,[2] Salesforce,[3] and SAP.[4] Next, we give more details of artifact-centric information systems by introducing four typical examples [39].

- *Enterprise Resource Planning (ERP) systems*. These systems encompass modules supporting business areas such as planning, manufacturing, sales, accounting, financial, human resource management, project management, inventory management and transportation [120]. In this way, an integrated and continuously updated view of core business processes is provided using

[1] https://dynamics.microsoft.com/.
[2] https://www.oracle.com/applications/erp.
[3] https://www.salesforce.com.
[4] https://www.sap.com/products/erp.html.

common databases maintained by a database management system. The two most important processes that most ERP systems fully cover are the procure-to-pay and the order-to-cash processes.

- *Customer Relationship Management (CRM) systems*. These systems manage a company's interaction with current and potential customers. They document the data from a range of different communication channels, including the company's website, telephone, email, live chat, marketing materials and more recently, social media. By analyzing such data, the company learns more about their target audiences and how to best cater to their needs, ultimately driving sales growth. In addition to individual customer relations, on a higher (aggregated) level, CRM systems also support sales and marketing activities related to products, pricing, distribution, and campaigning. Important processes supported by CRM systems are campaign-to-leads and lead-to-order.
- *Supply Chain Management (SCM) systems*. These systems enable enterprises to source the raw materials or components needed to create a product or service and deliver that product or service to customers. In other words, they focus on the support of logistics operations that integrate with suppliers and customers. On an operational level, SCM systems support the management of transportation, inbound and outbound warehousing, and inventory, as well as corresponding planning and calculation processes. On a technical level, SCM systems support electronic data interchange with suppliers and customers, as well as various tracking technologies such as Radio-Frequency Identification (RFID) and barcode scanning. Key supply chain processes are order-to-delivery and return-to-refund.
- *Product Lifecycle Management (PLM) systems*. The product lifecycle is the series of stages that every product passes through from conception to retirement.
  which the product is specified, designed, and validated; (ii) the realization phase, in which the manufacturing system is planned and actual products are built, assembled, and tested; and (iii) the service phase, in which products are sold and delivered, used, maintained, and eventually disposed of (i.e., having no new investment in new releases). PLM systems support the various processes of the lifecycle of a product from an engineering perspective. Important processes supported by PLM systems are idea-to-launch and different types of order processes including built-to-order, engineered-to-order, or assembled-to-order.

The systems explained above are the most widely employed artifact-centric information systems in practice. Artifact-centric information systems are domain-

specific process-aware information systems [39]. In other words, each type provides some fixed functions to cater to a specific domain, e.g., CRM systems focus on managing customer relations. Note that functions provided by different types can be overlapped. For instance, ERP systems may have functions provided by CRM, SCM and PLM, which are integrated into ERP systems. In this case, the integrated systems can be considered as a specific module (category) of ERP systems, e.g., ERP CRM modules.

1.2.3 Features of Artifact-Centric Information Systems

Artifact-centric information systems are different from case-centric information systems and have their own features. In this part, we first give more details about the modules (artifacts) comprising an artifact-centric information system and then discuss its features through a comparison with case-centric information systems. Since ERP is dominating in artifact-centric information systems (and sometimes it can also refer to other artifact-centric information systems), we use ERP to represent the artifact-centric information systems for the comparison. Similarly, WFM is used to represent the case-centric information systems.

ERP consists of modules which provide various functions. Unlike traditional in-house-designed company-specific systems, ERP systems integrate standard business process modules (like "add-ons") and support tailoring and adding modules based on the requirements of customers, as shown in Figure 1.9(b). These modules include manufacturing, accounting, HR, inventory management, etc. Besides, ERP allows free flow of information between modules and supports tracking data (transactions) across these modules. In summary, the architecture of ERP systems facilitates transparent integration of modules, and provides flow of information between all functions within the enterprise in a customizable and consistently visible manner.

Each module has a hard-coded implicit process (life-cycle). The hard-coded process means that (i) the functions (activities) provided by the module are fixed, i.e., users cannot define new functions, and (ii) the ordering of activities in the process is fixed, e.g., activity "validate order" can only happen after activity "create order" (i.e., an order can only be validated after it is created). In practice, one can customize the modules based on their needs. Additionally, each module provides parameters for users to configure. After module selection and parameter configuration, the business process is implicitly embedded in the selected modules and the parameter tables [22].

Table 1.2 lists some typical modules in ERP systems [19]. For instance, the "Order Management" category contains two modules "Order Processing" and

"Financial Processing". Actually, the category, e.g., "Order Management", can be considered as a module on a higher level. To achieve better "fit" between the prefabricated modules and the needs of the organization, ERP systems must be configured by selecting proper modules and setting various parameters. The more parameters an ERP system has, the more flexibility users have in configuring the business process.

Application Category	Typical Modules
Manufacturing Management	• Material Requirements Planning • Shop Floor Control • Master Production Scheduling • Bill of Materials
Order Management	• Order Processing • Financial Processing
Project Management	• Document Control • Project Accounting • Gantt Charts/Project Tracking
Accounting	• Accounts Payable • Accounts Receivable • General Ledger • Payroll

Table 1.2: Typical modules in ERP systems.

After introducing ERP systems, we compare them with classical WFM systems, as shown in Figure 1.9. Indicated by Figure 1.9(a), a workflow model is designed in a WFM system to describe the business process. Then workflow instances are created to carry out the real transactions, that are recorded in the systems. In contrast, ERP systems do not have an explicit workflow model and support to customize business processes by simply selecting modules and configuring parameters. Transactions of different modules are recorded in a central database.

Through the introduction of ERP systems and the comparison with WFM systems, the features of ERP systems are summarized as follows:

- *Artifact-centric*. Different from WFM systems, which are case-centric and focus on the control-flow, ERP systems are data-centric and focus on integrating internal and external information across business functions. We can use the analogy with programming languages to illustrate this difference. WFM systems are like procedure-oriented programming languages which

focus on the procedure of organizing codes to realize a goal. ERP systems are like the object-oriented programming languages. A module of an ERP system has functions and parameters just like a class that has methods and parameters, and the system focuses on how to integrate the methods provided by each class to realize a goal.

- *Complex interactions*. In a WFM system, the workflow model can be executed for any number of workflow instances. Note that although many instances can be handled in parallel, from the viewpoint of the WFM system

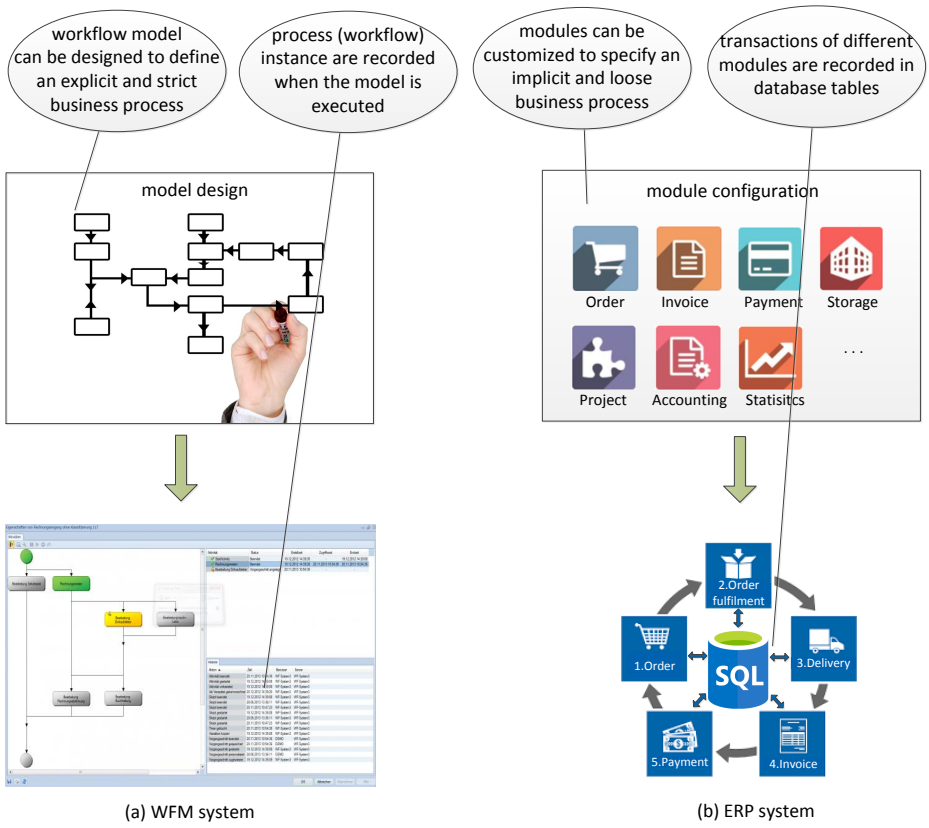


Figure 1.9: WFM systems versus ERP systems.

these instances are logically independent. In contrast, there exist complex interactions, such as one-to-many and many-to-many relations between instances of different artifacts. An ERP system provides a central server to store transactions of different modules, such that different modules can share data and interact with each other. It is possible that one instance of an artifact corresponds to multiple instances of another artifact and vice versa.

- *Flexible constraints*. In an ERP system, there is no explicit workflow model to specify the whole business process. Often the constraints in the implicit process are very loose (especially between different modules). In practice, although there exists a recommended business process, one can violate the process (if it is not hard-coded) when it is necessary. For instance, often an invoice is sent to the customer after its corresponding order. However, in real ERP systems such as Dolibarr, the invoice can be created anytime even without a corresponding order.

Next, we give a concrete example to explain the features of artifact-centric information systems. Figure 1.10 presents some events (sorted by time) of activities involved in the order-to-cash (OTC) scenario in an ERP system Dolibarr. These activities correspond to functions offered by different artifacts consisting of the ERP system. More precisely, activity "create_order" is a function provided by the "Commercial" artifact, which is used to create orders. Activities "create_invoice" and "create_payment" are functions of the "Financial" artifact. Activity "create_shipment" is from the "Product" artifact.

In Figure 1.10, events of different activities are represented by squares in different colors, e.g., the red square "create_order_1" (the first square on the left) means a "create order" event which happened at 10:33:37 on August 11th, 2017. Edges between squares connect related events. For instance, the edge connecting events "create_order_1" and "create_shipment_1" means that "create_shipment_1" shipped goods ordered by "create_order_1".

The complex relations between artifacts can be reflected by events of activities corresponding to these artifacts. For instance, there exists a many-to-many relation between activities "create order" and "create invoice". The "create order" event "create_order_1" corresponds to two "create invoice" events "create_invoice_1" and "create_invoice_2", and the "create invoice" event "create_invoice_2" is related to two "create order" events "create_order_1" and "create_order_2".

Note that conventional case-centric process mining approaches do not work properly on the event data shown in Figure 1.10. The XES log format can be considered as a process mining approach to organize events for further analysis. Here, we use it as an example to illustrate the problems. Figure 1.11 presents

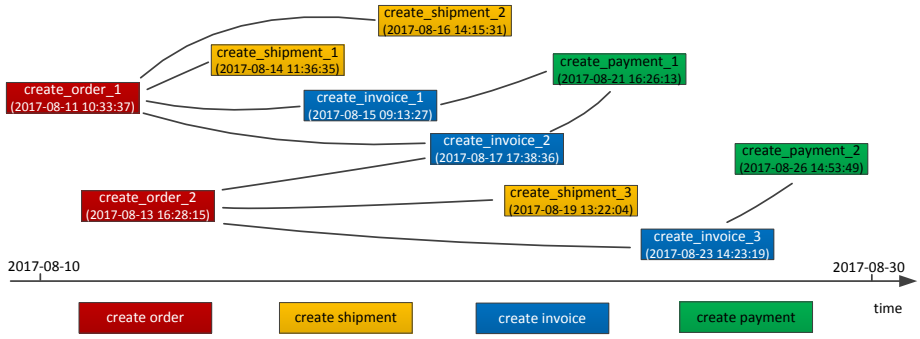


Figure 1.10: Events generated by artifact-centric information systems with one-to-many and many-to-many relations.

the XES log (with two cases $o1$ and $o2$), generated by flattening and splitting events in Figure 1.10, based on a straightjacket case notion, i.e., "order". Due to one-to-many and many-to-many relations, the generation of the XES log suffers two well-known problems: data divergence and data convergence.

Data convergence corresponds to the situation that one event refers to multiple cases. For instance, the event "create_invoice_2" is related to two orders created by "create_order_1" and "create_order_2". Considering "order" as the case notion, "create_invoice_2" is duplicated and split into two cases $o1$ and $o2$. The data convergence harms the log quality by leading to wrong frequencies of events because of event duplication. Data divergence means that a case is referred to by multiple events of the same activity. For instance, there are two events ("create_invoice_2" and"create_invoice_3") of the "create invoice" activity and two events ("create_payment_1" and "create_payment_2") of the "create payment" activity in the case $o2$. Due to the multiple instances of the same activity, the correspondences between different instances are misleading. For instance, in the case $o2$ it is not clear if "create_payment_2" is related to "create_invoice_2", as it is possible that (i) "create_payment_2" only pays for "create_invoice_3" or (ii) "create_payment_2" also pays for a part of "create_invoice_2" if "create_payment_1" does not pay for the whole "create_invoice_3".

This section illustrates the features of artifact-centric information systems. Based on a concrete example of data generated by such systems, it is revealed that existing process mining techniques may suffer convergence and divergence problems when dealing with these data. In next section, we will give more details about the possible problems.

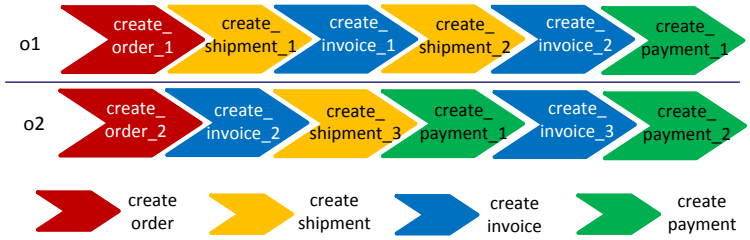


Figure 1.11: The XES log generated by flattening and splitting events in Figure 1.10 by choosing "order" as the case notion, which confronts convergence and divergence problems.

1.3 Challenges Encountered when Applying Process Mining to Artifact-Centric Information Systems

Process mining contains a set of technologies, which can analyze the data of PAISs from different angles. Existing process mining techniques make the fundamental assumption that process models and event logs are centered around a single case notion. However, as explained in Section 1.2.3, artifact-centric information systems such as ERP systems do not assume case notions in their business processes. Therefore, if we apply existing process mining techniques on data generated by these systems, they suffer the following problems:

1. It is *difficult (or impossible) to identify the case notion for the whole process*. Because of the artifact-centric nature of the data and processes, process instances are scattered over different departments in enterprises. The *case notion* varies in different departments. For instance, in the eyes of the "sales" department, the case notion is "order", whereas in the "delivery" department, the case notion is "shipment". Therefore, a global case notion for the whole process is missing.

2. It leads to *convergence and divergence problems*. There often exist one-to-many and many-to-many relations in the data generated by artifact-centric systems. If we straightjacket such data into XES logs (cf. Table 1.1), the original data with one-to-many and many-to-many relations is flattened as separate traces, in which one event cannot be referred to by multiple cases. This forced transformation compresses the data and leads to problems such as convergence and divergence (e.g., events are duplicated and multiple

instances of the same activity cannot be distinguished in one trace).

3. It *decreases the data quality*. An XES log consists of separate cases, which are recorded in isolation. Therefore, the interactions between events over different cases are missing. Besides, the XES format focuses on the behavioral perspective with considering other information as attributes attached to events. This imbalance weakens the data perspective of the original data.

4. It is *difficult to model interactions between process instances*. Existing process modeling languages consider process instances in isolation. Concepts like lanes, pools, and message flows in conventional languages like BPMN aim to address this. However, within each (sub)process a single instance is modeled in isolation.

5. It is *difficult to model the data-perspective and control-flow perspective in a unified and integrated manner*. Data objects can be modeled, but the more powerful constructs present in Entity Relationship (ER) models and UML class models, which can easily deal with many-to-many and one-to-many relationships, cannot be expressed well in process models. For example, cardinality constraints in the data model *must* influence behavior, but this is not reflected at all in today's process models.

6. It is *difficult to detect deviations related to the data perspective*. Existing conformance checking techniques mainly detect the deviations on the behavioral perspective. The violated constraints on the data perspective are often ignored. Besides, some deviating behavior keep undetected using existing approaches, since they can only be detected when taking into consideration the data perspective.

7. Performance analysis is *imprecise*. Existing performance analysis usually take XES logs as input. As mentioned above, the extracted XES logs from data generated by artifact-centric information systems often suffer convergence and divergence problems, which lead to imprecise performance analysis.

The above problems were not addressed by existing process mining techniques. This limits their usage on artifact-centric information systems. Therefore, we propose new process mining techniques to solve these problems.

1.4 Thesis Approach

In this thesis, we apply process mining techniques to the data generated by artifact-centric information systems. Since the existing techniques have limitations to deal with such data, we propose a series of novel process mining techniques to solve the problems discussed in Section 1.3.

Figure 1.12 shows the overall framework proposed in this thesis [86]. In general, the spectrum of our approaches covers all types of conventional process mining research, i.e., extracting event logs from execution data, discovering models from event logs, and checking conformance and analyzing performance based on logs and models. Our major contributions include a novel log format (with respect to XES/MXML log format) to organize data, a novel modeling language (with respect to Petri net, BPMN, etc.) to describe business processes, and related approaches such as model discovery, conformance checking and performance analysis. Based on the contributions, insights are provided to the stakeholders of business processes by analyzing data from artifact-centric information systems.

eXtensible object-centric event log. We propose a novel log format named *eXtensible Object-Centric* (XOC) to organize the data generated by artifact-centric information systems, as shown in Figure 1.13. The main feature of the XOC log format is that the requirement of a single case notion is removed, since artifact-centric information systems do not have a clear case notion for the whole

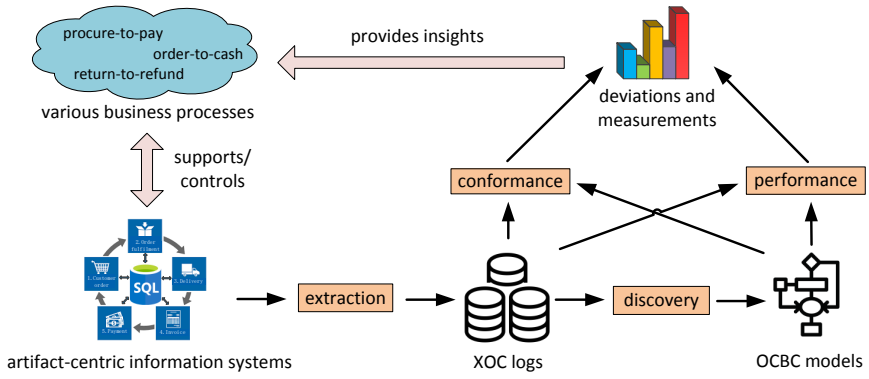


Figure 1.12: The framework relating the approaches proposed in this thesis.

process (**addressing problem 1**). In the execution of a business process on the artifact-centric information system, each event changes the state of the system, indicated by adding, updating or deleting records of the underlying database. Accordingly, an XOC log is specified as a sequence of events and each event corresponds to an object model (including rich information from the database), describing the state of the system (**addressing problem 3**).

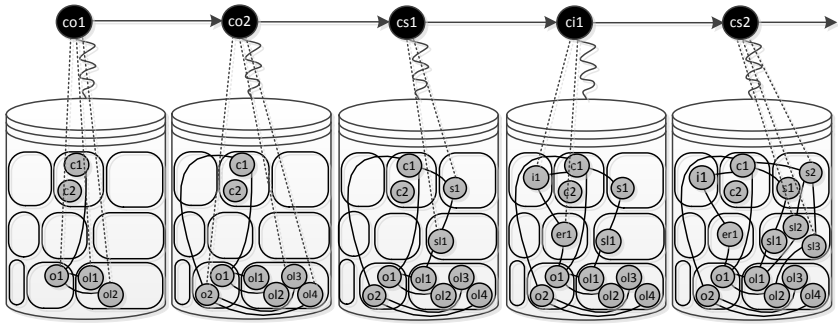


Figure 1.13: A "graphical" representation for an XOC event log, revealing the evolution of the state of an information system.

Event log extraction. An approach is proposed to automatically extract XOC logs from data generated by artifact-centric information systems. Since we do not add a forced case notion on the data in the transformation, the convergence and divergence problems are avoided (**addressing problem 2**).

Object-centric behavioral constraint modeling language. By combining data/object modeling languages (ER, UML, or ORM) and declarative languages (Declare), a novel modeling language, named *Object-Centric Behavioral Constraint* (OCBC), is proposed. More precisely, an OCBC model consists of a class model (presenting cardinality constraints between objects), a behavioral model (presenting declarative constraints between events) and so-called *AOC relationships* which connect these two models by relating activities in the behavioral model to object classes in the class model, as shown in Figure 1.14 (**addressing problems 4 and 5**).

Model discovery. We propose approaches to automatically discover OCBC models from XOC logs. Since the XOC logs have events and object models, it is possible to discover the data perspective and behavioral perspective, and the interactions in between (**addressing problems 4 and 5**).

Conformance checking. Based on an XOC log and a manually designed or discovered OCBC model, we propose a set of rules to check the conformance

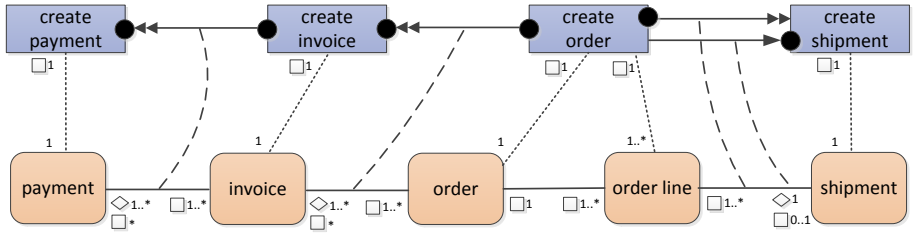


Figure 1.14: An *Object-Centric Behavioral Constraint* (OCBC) model example.

between them. The diagnostic results (i.e., deviations related to the behavioral perspective, data perspective and interactions) are present in three different views: rule view, log view and model view. Besides, metrics such as fitness, precision and generalization are defined to quantify the degree of conformance (**addressing problem 6**).

Performance analysis. By replaying an XOC log on an OCBC model, we can get the statistics of frequencies and time, and calculate some metrics based on the statistics. After mapping the performance result on the model, bottlenecks can be revealed. Since the input log does not have convergence and divergence problems, the performance result is precise (**addressing problem 7**).

1.5 Thesis Structure

We conclude the introduction with an outline of the thesis. This structure organizes all techniques involved in Figure 1.12 in a logical order.

After this introduction, Chapter 2 provides the preliminaries for this thesis. We first give the basic notations, which serve as a basis for the formal definitions in the thesis. Then, the necessary knowledge on the data perspective, such as data modeling languages and relational databases are introduced. Next, we illustrate some process modeling languages (such as Petri nets, BPMN diagrams and Declare models) and event logs.

Chapter 3 defines the notion of object-centric event data, to organize the data generated by artifact-centric information systems. Object-centric event data are different from case-centric event data and have their own characteristics. Accordingly, Chapter 4 proposes a novel log format, i.e., the eXtensible Object-Centric (XOC) log format to cater to object-centric event data.

Chapter 5 illustrates the novel OCBC modeling language by explaining all the involved elements. The data perspective (e.g., classes) is first explained, followed by the behavioral perspective (e.g., activities). Then these two perspectives are combined, resulting in an OCBC model.

Chapter 6 presents approaches to discover OCBC models from XOC logs. Without case notions, these approaches use the data perspective to correlate events and discover constraints. First, a basic approach is illustrated to discover models from clean logs. Then, a robust discovery approach is proposed to deal with noise in real life data.

Chapter 7 shows techniques to diagnose the conformance of the business process by comparing an XOC log and an OCBC model. By taking the data perspective into consideration, it is possible to detect and diagnose a range of conformance problems that would have remained undetected using conventional approaches.

Chapter 8 focuses on analyzing performance, which also takes an XOC log and an OCBC model as input. The dot charts, column charts, and indicators are employed to show the performance from different angles.

Chapter 9 shows a case study to link all techniques in a consistent manner and prove that these techniques can be applied to real life data while Chapter 10 concludes the thesis and discusses the future work.

Chapter 2

Preliminaries

In this chapter, we introduce necessary preliminaries such as basic mathematical notations, data modeling languages, database systems, process modeling languages and events logs.

2.1 Basic Notations

This section explains the basic notations of sets, multisets, functions, sequences and tuples, which are used as a basis of definitions in this thesis. For instance, sets are used in Definition 2.6 (e.g., P is a finite set of places), multisets are used in Definition 2.6 (e.g., $M \in \mathbb{B}(P)$ is a *multiset* over P denoting the marking of the net), functions are used in Definition 2.8 (e.g., $act \in E \rightarrow \mathcal{U}_A$ maps events onto activities), sequences are used in Definition 2.9 (e.g., each case corresponds to a trace, which is a finite sequence of events $\sigma \in (\mathcal{U}_E)^*$) and tuples are used in Definition 2.6 (e.g., a *Petri net* is a tuple $N = (P, T, F)$).

Definition 2.1 (Set) *A set is an unordered collection of distinct objects, which are called elements of the set. We denote a finite set by listing its elements between braces, e.g., a set A with elements a, b and c is denoted as $A = \{a, b, c\}$ and a set B with elements b, c and d is denoted as $B = \{b, c, d\}$.*

- $\in$ *indicates that an element is contained by a set, e.g., for the set A, $a \in A$ whereas $d \notin A$,*
- $\emptyset$ *denotes the empty set, i.e., $\emptyset = \{\,\}$,*

- $|X|$ *denotes the number of elements (*cardinality*) of a finite set X, e.g.,* $|A| = 3$,
- $X \cup Y$ *is the* union *of X and Y (i.e., the set of elements that exist either in X or Y), e.g.,* $A \cup B = \{a, b, c, d\}$,
- $X \cap Y$ *is the* intersection *of X and Y (i.e., the set of elements that exist in both X and Y), e.g.,* $A \cap B = \{b, c\}$,
- $X \setminus Y$ *is the* difference *of X and Y (i.e., the set of elements of X that do not exist in Y), e.g.,* $A \setminus B = \{a\}$ *and* $B \setminus A = \{d\}$,
- $X \subseteq Y$ *denotes that X is a* subset *of Y,*
- $X \subset Y$ *denotes that X is a* strict subset *of Y, and*
- $\mathscr{P}(X) = \{Y \mid Y \subseteq X\}$ *denotes the power set (i.e., the set of all subsets over X), e.g.,* $\mathscr{P}(A) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$.

A multiset modifies the concept of a set by allowing for multiple instances for each of its elements. The number of instances for each element is called the multiplicity of this element in the multiset.

Definition 2.2 (Multiset) *A multiset (bag) M is a tuple $M = (X, m)$ where X is a set and $m : X \to \mathbb{N}$ is the multiplicity function of the multiset. $x \in M$ denotes that element x is contained in the multiset M, i.e., $x \in X$ and $m(x) \geq 1$. $\mathbb{B}(X)$ denotes the set of all multisets over X and $[\,]$ denotes the empty multiset.*

Often, we use a compact notation for multisets. For example, we write $M = [a, a, b]$ or $M = [a^2, b]$ for the multiset $M = (\{a, b\}, m)$ with $m(a) = 2$ and $m(b) = 1$.

A multiset adds the information of multiplicity to each element in a set. However, it does not distinguish the order of elements. In a sequence, the position of each element matters, which is called the rank or index of the element.

Definition 2.3 (Sequence) *A sequence $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ is an ordered collection of objects, which is represented by listing its elements between angled brackets.*

- *σ_i refers to the i-th element of the sequence,*
- *$|\sigma| = n$ denotes the length of the sequence,*
- *$\langle\,\rangle$ denotes an empty sequence,*
- *A^* denotes the set of all finite sequences over a set A,*

- $\sigma_1 \oplus \sigma_2$ *appends sequence* σ_2 *to* σ_1 *resulting in a sequence of length* $|\sigma_1|+|\sigma_2|$*, e.g.,* $\langle a,b\rangle \oplus \langle c,d\rangle = \langle a,b,c,d\rangle$*,*
- ∂_{set} *converts the sequence* σ *into a set* $\{\sigma_1,\sigma_2,\ldots,\sigma_n\}$*, e.g.,* $\partial_{set}(\langle a,a,d\rangle) = \{a,d\}$*,*
- a *is an element of* σ*, denoted as* $a \in \sigma$*, if and only if* $a \in \partial_{set}(\sigma)$*,*
- $\partial_{multiset}$ *converts the sequence* σ *into a multiset* $[\sigma_1,\sigma_2,\ldots,\sigma_n]$*, e.g.,* $\partial_{multiset}(\langle a,a,d\rangle) = [a^2,d]$*, and*
- $tl^k(\sigma)$ *gets the "tail" of the sequence* σ *composed of the last* k *elements, e.g.,* $tl^2(\langle a,a,d\rangle) = \langle a,d\rangle$*.*

A function is a relation that uniquely associates each element of one set with a single element of another set. It is a many-to-one or one-to-one relation.

Definition 2.4 (Function) *Let* X *and* Y *be two non-empty sets. A function* f *from* X *to* Y *is a relation from* X *to* Y*, where every element of* X *is mapped onto an element of* Y *by* f*, denoted as* $f : X \to Y$*. It is denoted as* $f(x) = y$ *that* f *maps an element* $x \in X$ *onto an element* $y \in Y$*. For each function* f*,*

- $dom(f)$ *denotes the* domain *of* f*,*
- $rng(f)$ *denotes the* range *of* f*,*
- f *is* injective *if every element in the domain is uniquely mapped onto an element in the range, i.e.,* $\forall x,x' \in X : f(x) = f(x') \Rightarrow x = x'$*,*
- f *is* surjective *if each element in the range can be obtained by applying the function on an element from the domain, i.e.,* $\forall y \in Y : \exists x \in X : f(x) = y$*, and*
- f *is* bijective *if it is both surjective and injective.*

A function f *can be partial, denoted as* $f : X \nrightarrow Y$*, if* $dom(f) \subseteq X$*, i.e., not every element in the domain has to be mapped onto an element in the range.*

A function defines a unique mapping from elements of a source set to elements of a target set, which only involves two sets. It is possible to define a mapping between elements of multiple sets.

Definition 2.5 (Product and Tuple) *Let* $A_1, A_2, \ldots, A_n$ *be* n *sets.* $A_1 \times A_2 \times \ldots \times A_n = \{(a_1,a_2,\ldots,a_n) \mid a_1 \in A_1 \wedge a_2 \in A_2 \wedge \ldots \wedge a_n \in A_n\}$ *denotes the cartesian* product *of* $A_1, A_2, \ldots, A_n$*. An element* $t = (a_1,a_2,\ldots,a_n)$ *is a* tuple*.*

2.2 Data Modeling

Information systems employed in organizations provide an abundance of data. In order to describe and analyze the data structure in which these data are

stored, many data modeling languages have been proposed. In Chapter 5 we propose the OCBC modeling language which describes both behavioral and data perspectives of business processes. On the data perspective, the language models data objects by using powerful constructs (e.g., cardinality constraints) present in entity-relationship data models and UML class diagrams, which are widely used by conceptual data modeling languages. Therefore, in this section we introduce these two categories of models, to lay a foundation for our modeling languages.

2.2.1 Entity-Relationship Models

The entity-relationship (ER) data model describes the data involved in a real-world enterprise in terms of entities and their relationships. It was originally proposed by Peter Chen in 1976 [25], and is still widely used today. In this section, we introduce the ER model and discuss how its features allow users to model a wide range of data [119].

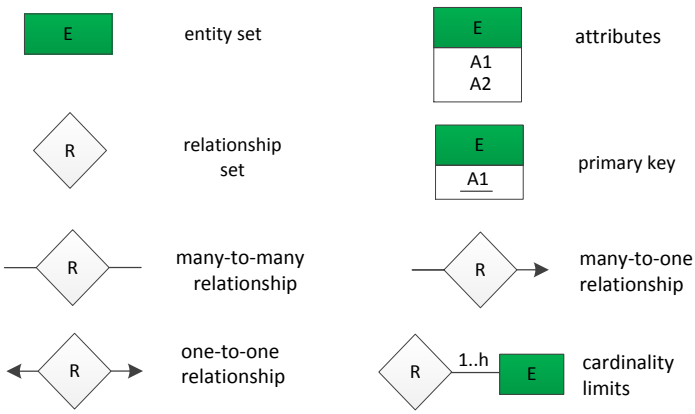


Figure 2.1: Example symbols used in the ER notation in [128].

A conceptual data model uses concepts such as entities, attributes and relationships to describe a data structure. More precisely, an entity represents a real-world object or concept, such as an employee or a project. An entity set is a set of entities of the same type that share the same properties, or attributes. An attribute represents some property of interest that further describes an entity, such as the name or salary of an employee. A relationship among two or more

entities represents an association among the entities, for example, a works-on relationship between an employee and a project [41]. A relationship set is a set of relationships of the same type.

An intuitive representation of the data model is significant for the communication between domain experts who know the requirements of the application but may not be familiar with data modeling. Note that there is no universal standard for ER model notations, and there exist different variants of notations. Figure 2.1 shows a particular variant in [128], and some typical notations are explained as follows:

- An entity set is represented by a square in green.
- A relationship set is represented by a diamond-shaped symbol.
- Attributes of an entity set are shown as annotations on the square representing the entity set.
- Attributes that are part of the primary key are underlined.
- Solid lines link entity sets to relationship sets (and the shape of the end of the line indicates the cardinality).
- A cardinality constraint can be added to the line between an entity set and a relationship set to indicate the cardinality limits.

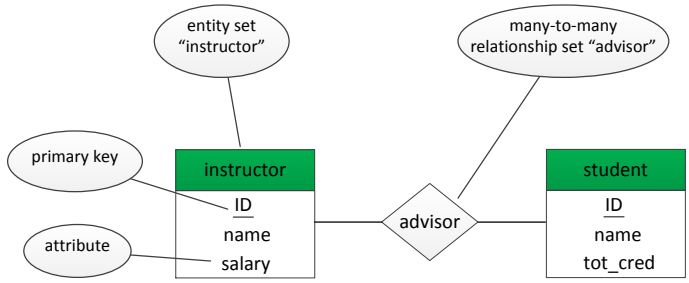


Figure 2.2: An ER diagram describing the relationship between instructors and students.

Consider the ER diagram in Figure 2.2, which consists of two entity sets, “instructor” and “student” related through a binary relationship set “advisor” [128]. The attributes associated with “instructor” are “ID”, “name” and “salary”, and the attributes associated with “student” are “ID”, “name” and “tot_cred”. Note that the attribute “ID” is underlined, which indicates that it is a member of the primary key of the entity set. The undirected lines from the relationship set “advisor” to both entity sets “instructor” and “student” indicate that an instructor

may advise many students, and a student may have many advisors.

2.2.2 UML Class Diagrams

The Unified Modeling Language (UML) is a standard developed by the Object Management Group (OMG) for creating specifications of various components of a software system [11, 16, 42, 53]. It includes a set of diagrams, such as class diagrams, use case diagrams, activity diagrams and implementation diagrams. The class diagrams relate to data modeling and they are similar to ER diagrams. In this part we introduce class diagrams and illustrate a few features of them.

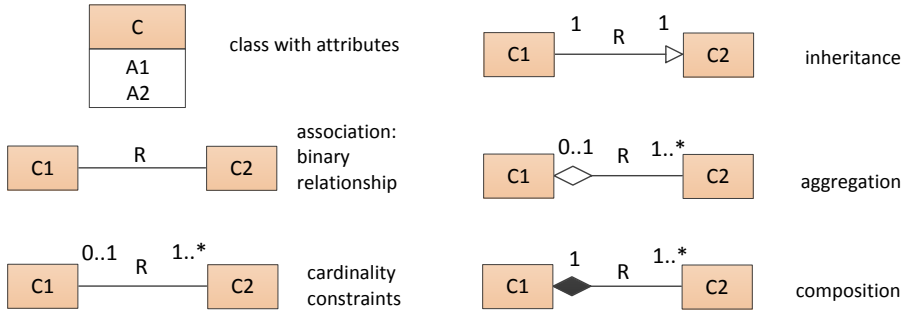


Figure 2.3: Symbols used in the UML class diagram notation.

In UML terminology, classes are used to describe entities and relationship sets are referred to as associations. A *class* is represented by a square in orange. *Attributes* of a class are shown as annotations on the square representing the class. An *association* in UML is represented by a solid line connecting the classes. The association name is adjacent to the line. *Cardinality constraints* are shown on the two sides of the line depicting the association [128]. Associations are of different types, which are explained as follows:

- *Inheritance*. Inheritance refers to the process of a child or sub-class taking on the functionality of a parent or superclass. It is symbolized as a line with a closed arrow (of a hollow triangle shape) pointing towards the superclass. For instance, in Figure 2.3 $C1$ can be a class of "car" and $C2$ can be a class of "vehicle" (i.e., $C1$ is the sub-class of $C2$ since "car" is a sub-class of "vehicle").
- *Aggregation*. An aggregation relationship means a part-whole relationship. More precisely, aggregation can occur when a ("whole") class is a collection

or container of other contained ("part") classes, but the contained classes do not have a strong lifecycle dependency on the container. In other words, the contents of the container still exist when the container is destroyed. As shown in Figure 2.3, an aggregation relationship is graphically represented as a hollow diamond shape on the containing class with a single line that connects it to the contained class. For instance, $C1$ can be a class of "library" and $C2$ can be a class of "book", i.e., a book is a part of the library and the book can exist without the library.

- *Composition*. A composition relationship shows an "entirely made of" relationship, i.e., a stronger version of aggregation. More precisely, composition can occur when a class is a collection or container of other contained classes, and the contained classes have a strong lifecycle dependency on the container. When the container is destroyed, the contents are also destroyed. As shown in Figure 2.3, a composition relationship is graphically depicted as a filled diamond shape on the containing class end of the line that connects the contained class to the containing class. For instance, $C1$ can be a class of "university" and $C2$ can be a class of "department", i.e., a department is a part of the university and the department cannot exist when the university is destroyed.

The data perspective of an OCBC model refers to the idea from UML class model. We also use the notion of classes and add attributes to classes. Differently, we only consider the binary relation with cardinality constraints, without distinguishing the different types. We add the "always" and "eventually" types of cardinality to strengthen the cardinalities. In future, it is possible to add new relations, such as inheritance.

2.3 Database Systems

The data analyzed by techniques proposed in this thesis are mainly from database systems. Therefore, in this section, we introduce database systems and discuss the key database concepts.

A database management system (DBMS) is a generalized software system for managing databases [41, 119, 128]. Database systems are designed to manage large bodies of information. The primary goal of a DBMS is to define structures to store and retrieve database information both conveniently and efficiently. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

Database systems are widely employed by modern information systems (and

in particular by artifact-centric information systems) to store and organize their execution data. Some well-known examples are Oracle RDBMS (Oracle), SQL server (Microsoft), PostgreSQL (PostgreSQL Global Development Group), DB2 (IBM) and Sybase (SAP).

2.3.1 Relational Models

Along with the development of computer techniques, information systems are widely employed by enterprises to store and manipulate the generated data. In early days, data were stored as files supported by file-processing systems. More precisely, the data were stored in independent files which do not communicate with each other. The systems which access the data use ad-hoc approaches (i.e., application-specific codes) to extract records from and add records to the appropriate files. This solution has a lot of problems when the data are large and complex, which drives the need for more efficient ways to store and manipulate data.

As an alternative to file-processing systems, databases appeared as a solution which can more efficiently store data. The first general-purpose DBMS was designed by Charles Bachman at General Electric in the early 1960s and was called the Integrated Data Store. In 1970, Edgar Codd proposed the relational data model at IBM's San Jose Research Laboratory [27]. This proved to be a watershed in the development of database systems: it sparked the rapid development of several DBMSs based on the relational model. Prototype relational database management systems were developed in pioneering research projects at IBM and UC-Berkeley by the mid-70s [119].

Today, the relational model is still by far the most dominant data model and is the foundation for the leading DBMS products, including IBM's DB2 family, Informix, Oracle, Sybase, Microsoft's Access and SQLServer, FoxBase and Paradox. Relational database systems are ubiquitous. The SQL query language for relational databases, developed as part of IBM's System R project, is now the standard query language. SQL was standardized in the late 1980s and since decades it is the most widely used language for creating, manipulating, and querying relational DBMSs [119].

In recent years, there has been an uptake of alternative databases, e.g., NoSQL databases and document oriented databases. NoSQL databases are often very fast, do not require fixed table schemas. Examples of NoSQL systems include MongoDB and Oracle NoSQL Database. In this thesis, we focus on relational databases since most artifact-centric information systems are based on them.

2.3.2 Tables

A relational database consists of a collection of tables, each of which is assigned a unique name. Each table has multiple columns and each column has a unique name. Besides, each table consists of rows which are used to represent both data and the relationships among those data.

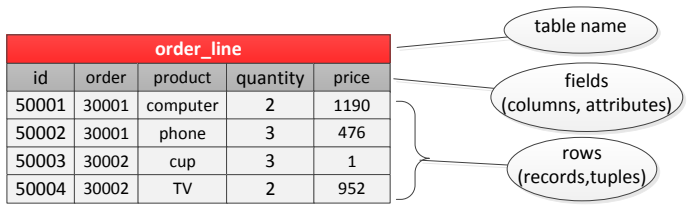


Figure 2.4: An example table.

Figure 2.4 shows a table named “order_line”, which stores information about details of customer orders in the OTC business process. The table has five columns (attributes, fields): “id”, “order”, “product”, “quantity” and “price”. Each row (tuple, record) of this table records information about an order line, i.e., the id of the order line, the order which the order line belongs to, the name, price and quantity of product contained by the order line. For instance, the first row corresponds to an order line “50001” which belongs to the order “30001”. It contains two computers whose price is “1190”.

The rows explained above are database instances at a particular moment, which represents a state of the database. In contrast, the overall design of the database (e.g., the columns) is called the database schema. The database schema defines the structure of a database by assigning constraints on database instances. Here, we introduce two types of constraints. The first type consists of domain constraints. For each column of a table, there is a set of permitted values, called the domain of that column. For instance, the “quantity” column in Figure 2.4 requires values of “INTEGER”. The second type consists of key constraints. A (unique) key constraint is a statement that a set of columns of a table is a unique identifier for a record. It is possible that there are multiple keys in a table, and the term *primary key* is used to denote a candidate key that is chosen by the database designer as the principal means of identifying records within a table. Sometimes the information stored in a table is linked to the information stored in another table. In this case, the *foreign key* can be defined in one table to reference the primary key in another table, in order to correlate tables.

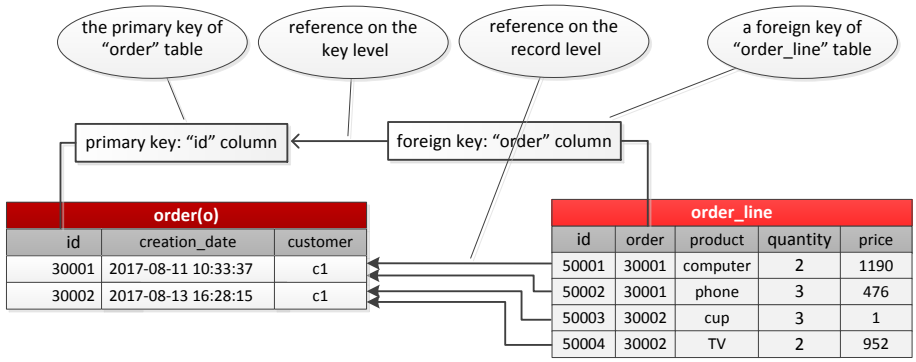


Figure 2.5: Primary keys and foreign keys in tables.

Figure 2.5 shows two related tables "order" and "order_line" (the records in the "order_line" table give the details of orders). The "id" columns in "order" table and "order_line" table have unique values and they can be considered as the primary keys of two tables. For instance, "30001" identifies the first record in "order" table. In order to connect records in these two tables, the "order" column can be considered as the foreign key of "order_line" table, which references the primary key of "order" table. For instance, the order line record "50001" references the order record "30001". Note that all referenced records must exist. For instance, the order line record "50004" references an order record "30002". Therefore, there must exist an order record "30002" in the "order" table.

2.4 Process Modeling

Information systems interact not only with database systems but also with the operational processes they support. A process can be defined as "the combination of a set of activities within an enterprise with a structure describing their logical order and dependence whose objective is to produce a desired result" [8]. Next to introducing database systems in Section 2.3, we illustrate process modeling languages in this section.

A large number of graphical process modeling languages has been proposed to describe the business processes by means of models, which facilitate human understanding and communication between the parties involved in the process (managers, analysts, modelers, etc.) [102]. In this section, we first introduce

Petri nets that are widely used in academia (especially in process mining domain) and BPMN diagrams that are widely used in industry. Both Petri nets and BPMN diagrams are "standards" in their own way, which are used to compare with OCBC models in this thesis. Then we explain Declare models as the behavioral perspective of the OCBC models refers to Declare models. At last, we give two examples of artifact-centric models, Proclet and GSM, which are used to describe artifact-centric information systems.

2.4.1 Petri Nets

Petri nets are the best-investigated process modeling language allowing for the modeling of concurrency. Petri nets use a very simple notation of circles representing places and squares representing transitions with arrows connecting them. Although the graphical notation is intuitive and simple, Petri nets are executable and many analysis techniques can be used to analyze them [71, 73, 74, 144, 162].

More precisely, a Petri net is a bipartite graph consisting of places and transitions. The network structure is static, but tokens (presented by black dots) can flow through the network, governed by the firing rule illustrated next. A transition can represent a task and when executed it consumes one token from each of its input places and produces a token in each of its output places. In this way, tokens are moved between places and the state of a Petri net is determined by the distribution of tokens over places. The distribution is referred to as the marking, and the initial (final) marking indicates the start (end) of the process.

Definition 2.6 (Petri net) *A* Petri net *is a tuple* $N = (P, T, F)$ *where* P *is a finite set of* places*,* T *is a finite set of* transitions *such that* $P \cap T = \emptyset$*, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the* flow relations*. A marked Petri net is a pair* (N, M)*, where* $N = (P, T, F)$ *is a Petri net and* $M \in \mathbb{B}(P)$ *is a* multiset *over* P *denoting the marking of the net.*

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F. Figure 2.6 shows an example of a Petri net, which describes the OTC business process. It can be formalized as follows: $P = \{p0, p1, p2, p3, p4, p5, p6, p7\}$, $T = \{a, b, c, d, e, t1, t2, t3\}$ and $F = \{(p0, a), (a, p1), \ldots, (b, p2)\}$. Note that $t1$ ($t2$ or $t3$) is a special transition, named silent, or τ-transition, denoted by a filled black square. A node x is an input node of another node y if and only if there is a directed arc from x to y (i.e., $(x, y) \in F$). For any $x \in P \cup T$, $\bullet x = \{y | (y, x) \in F\}$ denotes the set of input nodes and $x\bullet = \{y | (x, y) \in F\}$ denotes the set of output nodes. For instance, in Figure 2.6 $\bullet p1 = \{a\}$ and $p1\bullet = \{b, t1\}$.

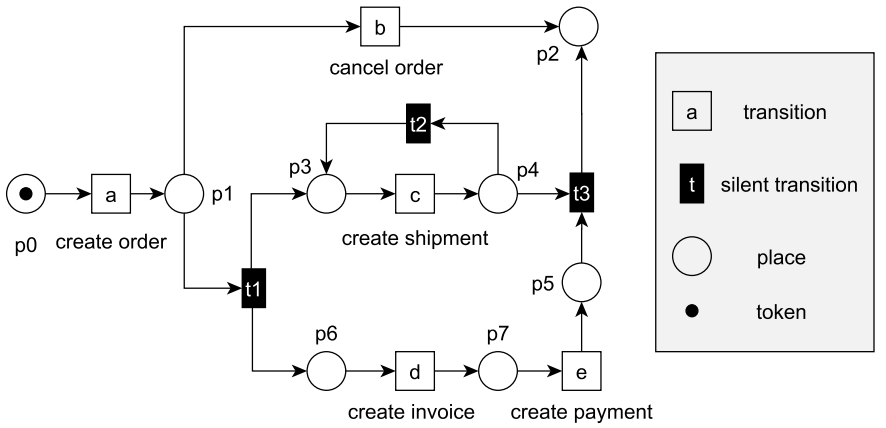


Figure 2.6: An example of a Petri net, describing the OTC business process.

The firing rule defines the dynamic behavior of a marked Petri net. A transition $t \in T$ is enabled in a marking M of net N, denoted as $(N,M)[t\rangle$ if each of its input places $\bullet t$ contains at least one token. For instance, in Figure 2.6 transition a is enabled since $p0$ (all of its input places) has one token, i.e., $(N,[p0])[a\rangle$. An enabled transition t may fire, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. $(N,M)[t\rangle(N,M')$ denotes that t is enabled in M and firing t results in the marking M'. For instance, in Figure 2.6, $(N,[p0])[a\rangle(N,[p1])$.

Let (N,M_0) with $N=(P,T,F)$ be a marked Petri net. A sequence $\sigma \in T^*$ is called a firing sequence of (N,M_0) if and only if, for some natural number $n \in \mathbb{N}$, there exist markings $M_1,M_2,\ldots,M_n$ and transitions $t_1,t_2,\ldots,t_n \in T$ such that $\sigma = \langle t_1,t_1,\ldots,t_n\rangle$ and, for all i with $0 \leq i < n$, $(N,M_i)[t_{i+1}\rangle$ and $(N,M_i)[t_{i+1}\rangle(N,M_{i+1})$. A marking M' is reachable from M if there exists a σ such that $(N,M)[\sigma\rangle(N,M')$. For instance, in the Petri net in Figure 2.6, $(N,[p0])[\sigma\rangle(N,[p2])$ for a sequence $\sigma = \langle a,b\rangle$.

Consider for example the Petri net in Figure 2.6 to understand how its described business process is executed. The initial marking is $[p0]$, which means that the model starts with a token in place $p0$. Based on the initial marking, the transition a is enabled. By firing transition a, the token in place $p0$ is consumed and a token is produced in place $p1$. Of the two transitions b and $t1$ only one can fire: they are in a so-called *exclusive choice* relation. If b is selected and fired,

the token in $p1$ is consumed and the final marking $[p2]$ is reached. Alternatively, if $t1$ is selected and fired, there are tokens in places $p3$ and $p6$, i.e., the marking is $[p3, p6]$, which enables two parallel branches c and d. Note that c can be fired more than once because of the loop construct involving c and a silent transaction $t2$, and firing c generates one token in $p4$. For the bottom branch, d and e are fired sequentially, resulting in a token in $p5$. The marking $[p4, p5]$ is derived after executing the two branches. Then the final marking $[p2]$ is reached after firing $t3$.

In the above explanation of Petri nets, transitions are identified by a single letter, which does not give information about the real business processes represented by Petri nets. Next, we define a labeled Petri net to relate transitions to activities, which can be derived based on the knowledge of the corresponding business processes.

Definition 2.7 (Labeled Petri net) *Let $\mathcal{U}_A$ denote the activity alphabet and τ denote a special (silent) label such that $\tau \notin \mathcal{U}_A$. A* labeled Petri net *is a tuple $PN = (P, T, F, l)$ where (P, T, F) is a Petri net and $l \in T \rightarrow \mathcal{U}_A \cup \{\tau\}$ is a labeling function .*

A labeled Petri net is a Petri net, in which each transition has a corresponding activity. For instance, the Petri net in Figure 2.6 is a labeled Petri net. Function l indicates the activity for each transition, e.g., $l(a) = \textit{create order}$. Note that it is possible that particular transitions are not observable. Such transitions are often referred to as silent or invisible, and we reserve the special activity label τ for them. For instance, transitions $t1$, $t2$ and $t3$ in Figure 2.6 are not observable. Therefore, $l(t1) = l(t2) = l(t3) = \tau$.

2.4.2 BPMN Diagrams

The *Business Process Modeling Notation* (BPMN) has become one of the most widely used languages to model business processes [55]. It provides a lot of constructs for representing in a very expressive graphical way and has enjoyed high levels of attention in business practice. The BPMN 1.0 specification was released in May 2004, and was adopted as a standard by the *Object Management Group* (OMG) in February 2006. From then on, it has quickly become a de-facto standard for graphical process modeling. It is widely supported by tool vendors (e.g., Activiti, Bonita BPM, Camunda, IBM Rational System Architect, and Signavio).

An example of a BPMN process model is shown in Figure 2.7, which describes the OTC business process. It is similar to the Petri net in Figure 2.6, and most

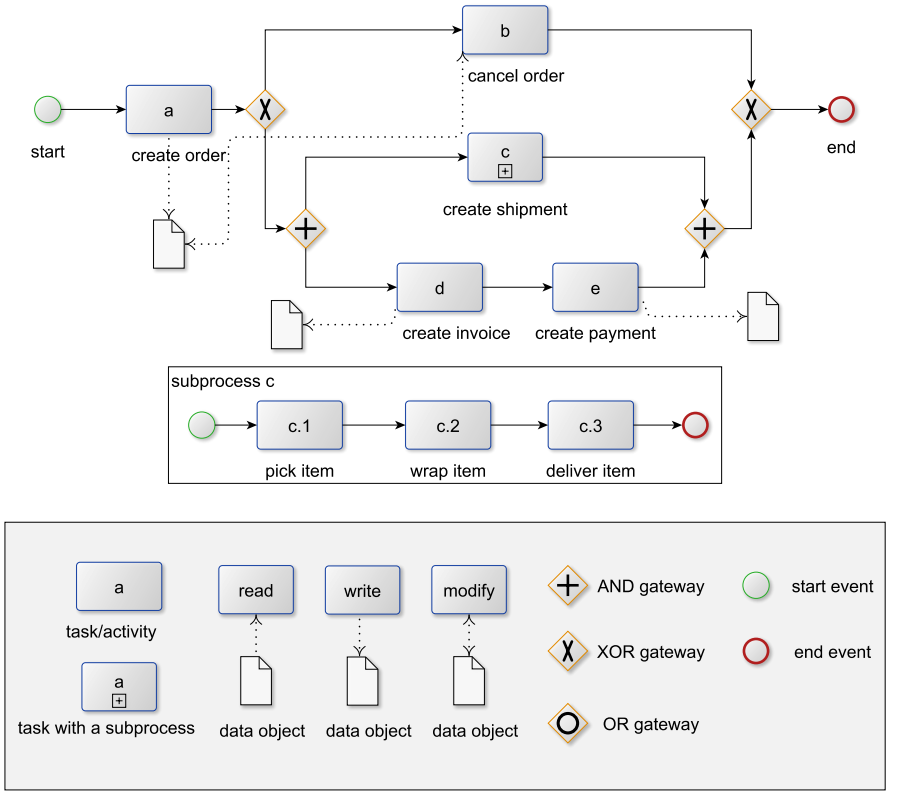


Figure 2.7: Describing the OTC business process using the BPMN notation.

constructs can easily be understood in the same way. In BPMN models, atomic activities are called tasks, represented by rounded rectangles. Note that the notion of composite tasks is modeled via a subprocess task, represented by a rounded rectangle with a "+" symbol. For instance, the task *c* (i.e., "create shipment") has a subprocess, consisting of tasks "pick item", "wrap item" and "deliver item". There are split and join gateways of different types: AND, XOR and OR. The BPMN process starts with a start event (circle) and ends with an end event (thick circle). Start events have one outgoing arc and end events have one incoming arc. Figure 2.7 shows just a subset of all notations provided by BPMN. Most vendors support only a small subset of BPMN in their products

since users typically use only a few BPMN constructs. The research performed by [166] has shown that typically less than 10 different symbols are used, while more than 50 distinct graphical elements are available.

BPMN initially focuses its attention on the control-flow perspective, i.e., describing which activities are performed and the dependencies between them, while the data perspective does not receive much attention. In order to strengthen the data perspective of BPMN, several changes have been made from BPMN 1.2 to BPMN 2.0. In BPMN 2.0 the data can be represented in a process diagram using the elements presented in Figure 2.8:

- Data objects. Data objects represent the information needed (data input) or produced (data output) by the activity, that flows through a process. A data object can be referenced by "DataObjectReference", which is a way to reuse one data object in the same diagram. A data object reference can

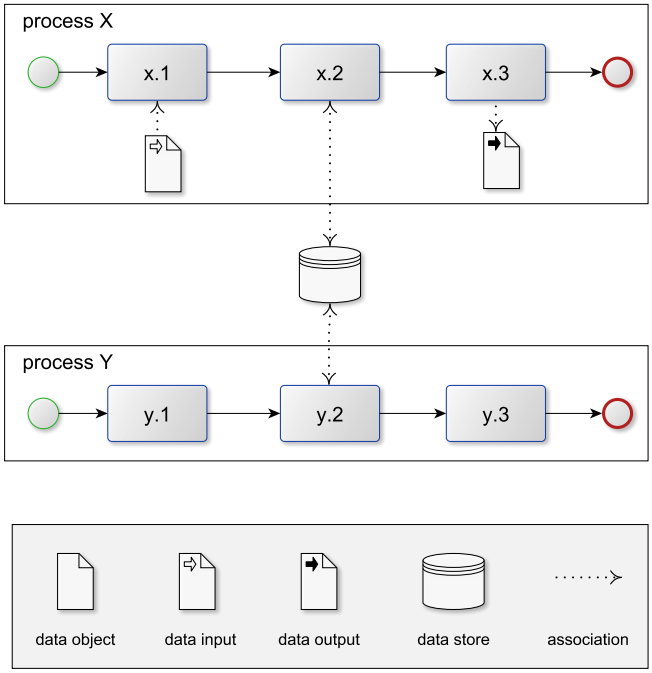


Figure 2.8: A BPMN model with data elements.

represent a different state of the same data object at different points in the process.

- Data stores. A data store is a means to handle persistent data. It provides a mechanism for an activity to store information or use the information stored. A data store can represent paper documents (a file folder, an agenda, a notebook, etc.) or a database.

In BPMN, data storage allows for interaction between different processes as shown in Figure 2.8, which is impossible with simple data objects, because they can only be used within one process.

2.4.3 Declare Models

Case-centric information systems often drive their business processes based on strict procedural process models, such as Petri nets, BPMN diagrams and EPCs. These models are suitable for describing standardized processes in stable environments. Because of their predictability and low complexity, these processes are described in a "closed world" manner, i.e., explicitly representing all the allowed behavior of the processes.

In order to deal with the fast-changing environments, artifact-centric information systems tend to provide more flexibility and less process-related support to match dynamic process management. Current procedural process modeling languages and models are of imperative nature, which forces designer to over-specify business processes. As a result, the systems based on such models suffer frequent re-engineering to adapt to rapid changes, which lead to inefficiency in enterprises. Besides, it also leads to complex and incomprehensible models using the procedural languages to describe processes supported by such systems.

In order to solve the problems introduced above, [113] proposes a declarative language, *Declare* (also called *ConDec*), to make a fundamental paradigm shift for flexible process management. *Declare* combines a formal semantics grounded in Linear Temporal Logic (LTL) on finite traces [142], with a graphical representation. In essence, a Declare model is a collection of LTL rules, each capturing a control-flow dependency between two activities [94]. In other words, a Declare model describes a set of constraints which must be satisfied throughout the process execution.

Declarative process modeling languages are more appropriate for artifact-centric information systems. Note that conventional procedural modeling languages produce "closed" models, i.e., all what is not explicitly specified is forbidden. In contrast, declarative models describe a process in an "open world"

manner, i.e., everything is allowed unless it is explicitly forbidden [94, 95]. The difference between procedural (imperative) and declarative modeling languages can be understood by analogy to programming language in computer programming. Imperative programming languages indicate "how to do something", whereas declarative programming languages indicate "what is required and let the system determine how to achieve it". Similarly, imperative process modeling primarily specifies the procedure of how work has to be done, i.e., they require all execution alternatives to be explicitly specified in the model during build-time before the execution of the process. It often results in process models being over-specified [112]. In contrast to imperative languages, declarative languages do not specify the procedure a priori. Instead of determining how the process has to work exactly, only its essential characteristics are described [114].

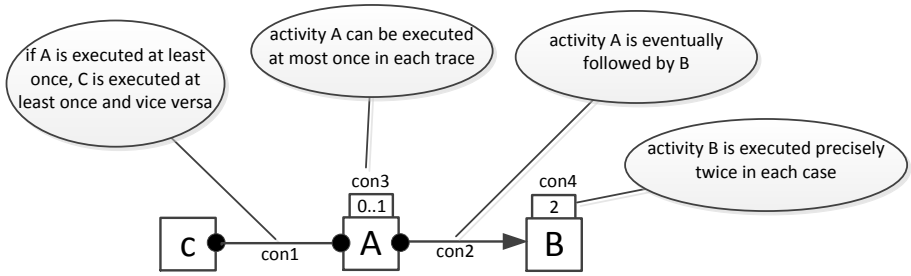


Figure 2.9: An example of a Declare model.

Figure 2.9 shows an example of a Declare model, which has three activities (A, B and C) and four constraints ($con1$, $con2$, $con3$ and $con4$). For instance, the constraint $con2$ between A and B requires that activity A is eventually followed by B. The constraints do not explicitly specify the flow of the interactions among process events. They can be understood as follows. Initially, the model only contains activities, allowing every possible execution behavior. By adding constraints to the model, execution alternatives are discarded step by step. For instance, if we only have three activities without any constraints in Figure 2.9, the activities can be executed arbitrarily often and in any order. After adding the constraint "0..1" onto activity A, some behavior is disabled, i.e., activity A is not allowed to happen more than once [114].

The constraints in Declare models depend on so-called *templates*, classified as existence templates, relation templates, negation templates, choice templates and branching templates. A template indicates the restriction assigned on activities involved in the template. Next, we illustrate two typical types, i.e., relation

templates and negation templates, which are referred to by the behavioral perspective of OCBC models.

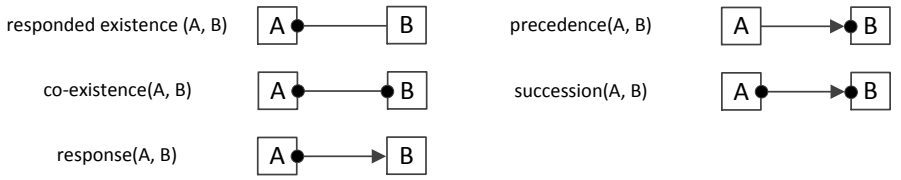


Figure 2.10: Notations for the relation templates.

A relation template defines a dependency between multiple activities. Figure 2.10 only shows some examples of binary relationships (i.e., between two activities). The *responded existence* template means that if activity A is executed, activity B also has to be executed either before or after activity A. According to the *co-existence* template, if one of the activities A or B happens, the other one has to happen as well. Different from the first two templates, the other three templates *response*, *precedence* and *succession* consider the ordering of activities. Template *response* requires that every time activity A executes, activity B has to be executed after it. The template *precedence* means that activity B can be executed only after activity A is executed. By combining *response* and *precedence* templates, the succession template is defined, which requires that both response and precedence relations have to hold between the activities A and B.

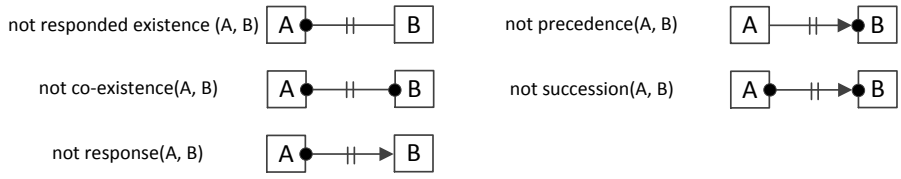


Figure 2.11: Notations for the negation templates.

Figure 2.11 presents some examples of negation templates, which are the negated versions of the relation templates in Figure 2.10. The *not responded existence* template means that if activity A happens, activity B can never happen (neither before nor after activity A). The *not co-existence* template indicates that if one of the activities A or B is executed, the other one must never be executed. The *not response* template specifies that after the execution of activity A, activity

B cannot be executed anymore. Required by the template *not precedence*, activity *B* cannot be preceded by activity *A*. The *not succession* template combines the *not response* and *not precedence* templates.

2.4.4 Artifact-Centric Models

Traditional languages to describe business processes, such as Petri nets, BPMN diagrams and EPCs, are case-centric, i.e., they assume a case notion in the process and often focus on the control-flow perspective (considering the data perspective as an afterthought). As we mentioned, artifact-centric systems do not assume a single case notion. Besides, such systems do not have explicit business processes but have clear business entities such as orders and invoices. As a result, it makes little sense (i.e., provides few insights) to only analyze the control-flow perspective.

In order to solve the above problem, artifact-centric [14, 28, 65, 107] (including the work on proclets [139, 140]) approaches were proposed. In this section, we first introduce Proclet models, which appeared earlier and share the same idea as artifacts. Then, GSM is explained as a notation to understand artifacts. At last, we discuss the problems confronted by these approaches.

Proclet Models

Most of today's business process modeling languages assume that a business process can be modeled by specifying the life-cycle of a single case in isolation. For many real-life applications this assumption is too restrictive, since business processes in these applications cannot be captured fully using a single case notion. As a result, the real process has to be adapted to accommodate these languages, i.e., the control-flow of several cases is artificially squeezed into one process definition.

For instance, in the OTC scenario, an order can include multiple order lines (items). Accordingly, there exist two processes on two different levels, the process executed for the whole order (e.g., sending invoices and delivering packages for the order) and the (sub)process executed for each order line (e.g., picking and wrapping items for each order line). Since an order can have a variable number of order lines and the order lines are processed concurrently, it is typically not possible to squeeze this scenario into one process definition. This is due to the fact that, in most WFM/BPM systems, the degree of parallelism is fixed in a process definition, i.e., it is not possible to concurrently instantiate selected parts (i.e., sub-processes) of the whole process a variable number of times. Although

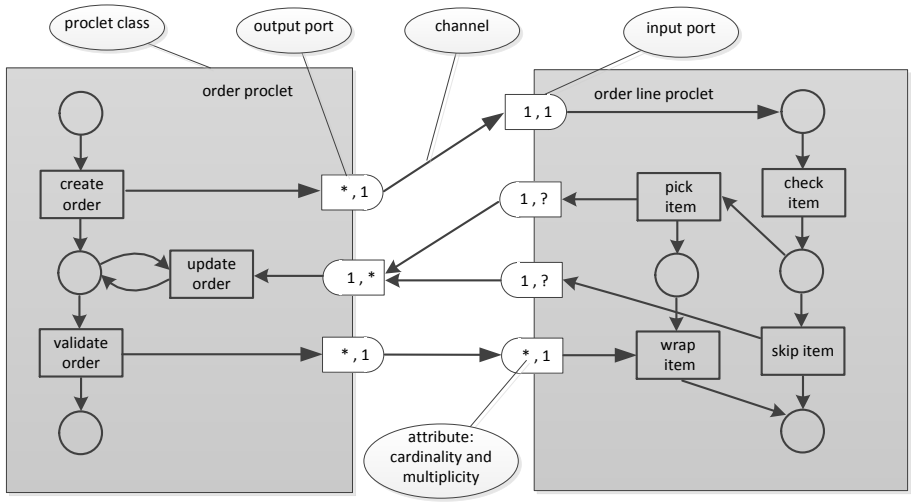
one can instantiate sub-processes a variable number of times through iteration or loop, it leads to the sequential execution of inherently parallel tasks.

To model the process discussed above, a framework based on *proclets* was proposed in [139, 140]. A proclet can be considered as a lightweight business process (process fragment, sub-process or smaller process) equipped with a knowledge base containing information on previous interactions. Based on this idea, the whole process including multiple sub-processes can be described as a proclet system, explained as follows:

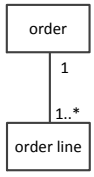
- *Proclet system*. A proclet system consists of proclet classes and interactions found between them.
- *Proclet*. A proclet class describes only one element or one perspective of the whole business process. For instance, a proclet class can describe the life-cycle (i.e. control-flow) of an artifact as a Petri net. Proclet instances can be created and destroyed, and are executed according to a class specification. At any moment a proclet instance has a state. When there is no confusion we can simply use the term "proclet" instead of "proclet class" and/or "proclet instance".
- *Channel*. Different proclet classes can interact via channels. A channel is the medium to send a uni-directional message, called performative, from one proclet class to another specific proclet class or a group of proclet classes. A channel connects two proclet classes through ports of the two proclet classes.
- *Port*. There exist two types of ports, i.e., input ports for receiving messages and output ports for sending messages. Each port has two attributes: (i) cardinality (at the left of the comma) and (ii) multiplicity (at the right of the comma).
- *Attribute*. The cardinality specifies the number of recipients of messages exchanged via the port. The multiplicity specifies the number of messages exchanged via the port during the lifetime of any instance of the proclet class.
- *Attribute value*. A value of a cardinality could be "1" (exactly once), "?" (zero or once), "$*$" (zero or more) and "+" (once or more).

Figure 2.12(a) describes the control-flow perspective of a set of collaborative processes in the OTC scenario. These processes can be described as two proclet classes, order and order line, between which there is a one-to-many relationship as indicated in Figure 2.12(b). In Figure 2.12(a), each square filled in grey represents a proclet class, and a Petri net is employed to model the lifecycle of the proclet class. On the edges of a proclet class, input ports (the round edge facing inside) and output ports (the round edge facing outside) are displayed.

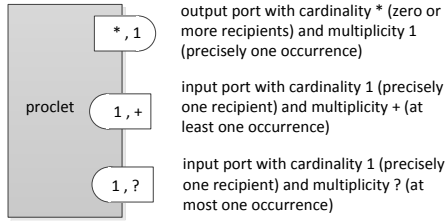
Channels are represented as edges from output ports to input ports. Attributes are shown on ports, and some examples are present in Figure 2.12(c).



(a) Two proclet classes describing the relation between orders and order lines.



(b) Class diagram of the two proclet classes.



(c) Examples of port attributes.

Figure 2.12: Example of two proclet classes: order and order line.

The semantics of proclets are illustrated next based on the two proclet classes described using Petri Nets (i.e., WF-nets) in Figure 2.12(a). Proclet class "order" is instantiated once per order while proclet class "order line" is instantiated for every specific item in the order. The instance of class order first multicasts a message to check if each item involved in the order exists in the warehouse. Note that the cardinality of the port connected to task "create order" is denoted as "$*$". This indicates that the order contains an arbitrary number of order lines.

The multiplicity of the same port is denoted by the number "1". This means that during the lifetime of an instance of "order", exactly one checking request is sent via this port. For each item, it is picked if it exists, otherwise it is skipped. The checking result is sent to the port connected to task "update order". Note that the multiplicities of the ports connected to tasks "pick item" and "skip item" are "?", since only one message is sent via one of the two ports (i.e., each port sends no or just one message). After receiving checking result, the order is updated (e.g., the item is removed from the order if it is out of stock). At last, the order is validated and a message is sent for wrapping the item.

Note that for the graphical representation of proclets, a selection can be made between multiple graphical languages, such as YAWL language, Petri Nets and EPCs. For instance, Petri nets are used in Figure 2.12(a) and the YAWL models are used in [100] to specify the proclets in healthcare.

GSM

Artifacts are key business-relevant entities/objects that are created, evolved, and (typically) archived as they pass through the business operations. They combine both the data perspective and the control-flow perspective into a holistic unit, and serve as the basic building blocks from which models of business operations and processes are constructed [24, 28]. An artifact type contains both an information model (the data perspective) that specifies all of the business-relevant data about entities of that type, and a lifecycle model (the control-flow perspective) that describes how events and activities affect the state of the artifact.

More precisely, an information model consists of *data attributes* and *status attributes*. Data attributes are intended to hold all business-relevant data about a given instance as it moves through the business process. Status attributes hold information about the current status by recording update time of related activities.

For the lifecycle model, the Guard-Stage-Milestone (GSM) meta-model [66, 67] provides a more declarative approach for modeling artifact lifecycles. It allows a natural way for representing hierarchy and parallelism within the same artifact and between different artifacts, and has become the most widely used notation to describe the lifecycles. The core three components of GSM models are, as the meta-model name suggests, stages, milestones and guards:

- *Stage*. A stage is a cluster of activities that might be performed for, with, and/or by an artifact instance, in order to achieve one of the milestones owned by that stage. A stage becomes "inactive" (or "closed") when one of its milestones is achieved. Intuitively, this is because the overall motivation

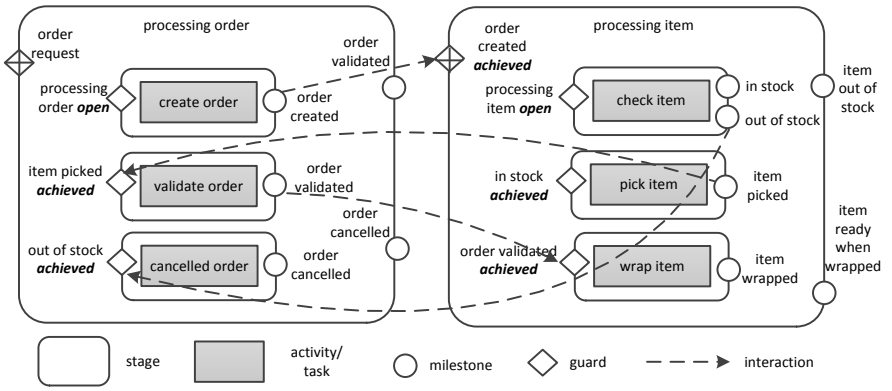
for executing a stage is to achieve one of its milestones and at most one milestone of a stage can be true at a time. Stages may be nested and complex stages contain one or more children stages. Atomic stages cannot have sub-stages, but are placeholders for tasks.

- *Milestone*. A milestone is a business-relevant operational objective (at different levels of granularity) that can be achieved by an artifact instance. A milestone corresponds to one alternative way that the stage might reach completion. It may be "achieved" (and become true when considered as a Boolean attribute) and may be "invalidated" (and become false when considered as a Boolean attribute). This information is recorded in corresponded status attribute in the information model as a Boolean attribute that indicates if the milestone is currently achieved. Information model also contains a status attribute that holds the timestamp when the milestone last changes.
- *Guard*. They are used to control whether a stage becomes "active" (or "open"). If a guard for a currently closed stage S becomes true (and the parent of S is already open if S has a parent stage), then S becomes open. There is also one special kind of guards, called bootstrapping guards, which specify the condition when the corresponding artifact instance is to be created (instantiated). Each artifact must have exactly one bootstrapping guard that can be associated with some top level stage.

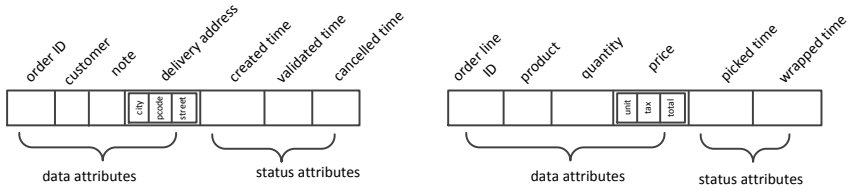
Besides, each guard has a form of an expression, called sentry. Strictly speaking, a guard is a sentry. Sentries are used as guards, to control when stages open, and to control when milestones are achieved or invalidated. A sentry consists of a triggering event type and/or a condition.

Figure 2.13 describes a part of the OTC business process with artifact-centric models. Here, we consider two typical artifacts, i.e., order and order line (or item). Figure 2.13(a) presents the lifecycle models and Figure 2.13(b) shows the information models of two artifacts. Consider the order artifact for example. Its information model includes slots for "order ID", "customer", "note", "delivery address", "created time", "validated time" and "canceled time". Its lifecycle model includes operations such as creating the order when receiving an order request, validating the order when all items exist in stock and cancelling the order if items are out of stock.

Note that the lifecycle models in Figure 2.13(a) employ the GSM notation. The rounded rectangles correspond to stages, e.g., the "processing order" stage on the left and the "processing item" stage on the right. The squares filled in grey represent activities or tasks, e.g., "create order". Guards are depicted by diamond nodes and bootstrapping guards have extra "+" symbols, e.g., "order



(a) Lifecycle models (using GSM notation) of order and order line (item) artifacts.



(b) Information models of order and order line artifacts.

Figure 2.13: Describing the OTC business process with artifact-centric models.

request". Milestones are shown as small circles associated with stages, e.g., "order validated". Note that there exist interactions between different artifacts, which are represented by dashed lines. For instance, the dashed line from the milestone "order created" to the guard "order created achieved" means that when the milestone is achieved, the guard is triggered.

Artifacts define a useful way to understand and track business operations. As shown in Figure 2.13, after an "order request", an instance of "order" artifact is created and the stage "processing order" is open. Then, the guard "processing order open" is triggered, and the activity "create order" is executed, which achieves the "order created" milestone. Through the interaction, the guard "order created achieved" is triggered, which creates an instance of "order line" artifact and opens the "processing item" stage. The guard "processing item open" is triggered and the corresponding item is checked. If the item is in stock, it is picked which triggers the guard "item picked achieved" in the "order" artifact.

Then the order is validated and the item is wrapped for delivery. If the item is out of stock, the guard "out of stock achieved" in the "order" artifact is triggered, which cancels the order.

To facilitate the application of the artifact-centric idea, a framework called BALSA (consisting of four dimensions, i.e., Business Artifacts, Lifecycles, Services and Associations) is proposed in [15, 65], which establishes the common ground for artifact-centric business process modeling. Employing different models and constructs in each of the four dimensions, one can obtain different artifact-centric business process models with differing characteristics. For instance, the data models used for artifacts might be focused on nested name-value pairs as in [107], nested relations or restricted Entity-Relationship schemas as in [28]. For the lifecyle models, besides Petri nets, other notations can be employed. Finite state machines have been traditionally used to model the individual lifecycles of artifacts, where each state of the machine corresponds to a possible stage in the lifecycle of an artifact. In this variant of state machines, little or nothing is indicated about why or how an artifact might move from one stage to another, although "guard" conditions may be attached to transitions in the machine [65].

The artifact-centric approaches have data and lifecycles attached to them, thus relating both behavioral and data perspectives. They do not require a single case notion for the whole business process. However, the relationships between artifacts are not made explicit. In summary, these approaches tend to have problems as follows:

- The data perspective (e.g., information models) presented by artifact-centric approaches has no direct relation to the mainstream data models, e.g., UML class models.
- Artifacts have to be identified beforehand based on domain knowledge and within an artifact (proclet), one is still forced to pick a single case notion. Assuming that one wants to discover the model in Figure 2.13 from database tables, he or she has to identify the order and order line artifacts based on some common knowledge. Without such knowledge, it is difficult to figure out artifacts and their corresponding tables. Besides, each artifact can involve multiple tables, and one has to choose a case notion at the artifact level to correlate events from different tables.

2.5 Event Logs

Process mining is impossible without proper event logs, which are the main input for most process mining techniques. An event log stores the execution

history of a process, which can be extracted from a variety of data sources, e.g., enterprise information systems, business transaction logs, web servers, databases and high-tech systems such as X-ray machines.

Table 2.1 shows an excerpt of an example data set used for traditional process mining. The data set stores events generated in a compensation request process in an airline. Each line in the table represents one event and each column represents an attribute of this event. More precisely, events need to be uniquely identified, which is achieved by assigning unique identifiers in the "Event id" column. Each event has a corresponding activity (a special attribute), indicated by the "Activity" column. For instance, the first line represents one "register request" event, with id "35654423", happening at "30-12-2010:11.02" and executed by Pete with a cost of 50.

Case id	Event id	Timestamp	Activity	Other attributes
1	35654423	30-12-2010:11.02	register request	Res:Pete, Cost:50
1	35654424	31-12-2010:10.06	examine thoroughly	Res:Sue, Cost:400
1	35654425	05-01-2011:15.12	check ticket	Res:Mike, Cost:100
1	35654426	06-01-2011:11.18	decide	Res:Sara, Cost:200
1	35654427	07-01-2011:14.24	reject request	Res:Pete, Cost:200
2	35654483	30-12-2010:11.32	register request	Res:Mike, Cost:50
2	35654485	30-12-2010:12.12	check ticket	Res:Mike, Cost:100
2	35654487	30-12-2010:14.16	examine casually	Res:Pete, Cost:400
2	35654488	05-01-2011:11.22	decide	Res:Sara, Cost:200
2	35654489	08-01-2011:12.05	pay compensation	Res:Ellen, Cost:200
...	...	...	...	...

Table 2.1: A segment of "case-centric" event data with a case notion.

In the example data set, each event is associated with a case. Unfortunately, not all information systems record events in this way (i.e., assuming a case notion). Information about the relation between events and cases is often not recorded in artifact-centric systems such as ERP/CRM. These systems do not have an explicit event log [45]. In order to conform to the data both from case-centric and artifact-centric systems, we next define a meta event log, and the conventional event log (e.g., an XES log) can be considered as a "subclass" of a meta event log. To recap, we assume the following general structure of a meta event log:

- An event log consists of events.
- An event corresponds to an activity.
- An event can have some attributes.
- Events within an event log are ordered.

Definition 2.8 (Meta Event Log) *Let $\mathcal{U}_E$ be the event universe, i.e., the set of all possible event identifiers, $\mathcal{U}_A$ be the activity universe, $\mathcal{U}_{Attr}$ be the universe of attribute names and $\mathcal{U}_{Val}$ be the universe of attribute values. A* meta event log *is a tuple $L = (E, act, attrE, \preceq)$, where*

- *$E \subseteq \mathcal{U}_E$ is a set of events,*
- *$act \in E \to \mathcal{U}_A$ maps events onto activities,*
- *$attrE \in E \to (\mathcal{U}_{Attr} \not\to \mathcal{U}_{Val})$ maps events onto a partial function assigning values to some attributes, and*
- *$\preceq \subseteq E \times E$ defines a total order on events.*[1]

The example data set is a "case-centric" data set. In other words, each event is associated with a case, or process instance. In Table 2.1 the events are already grouped by case and sorted chronologically. For instance, all the first five events correspond to case "1". The sequence of events that is recorded for a process instance is called a trace.

Definition 2.9 (Case and Trace) *Let $\mathcal{U}_{Cas}$ be the case universe, i.e., the set of all possible case identifiers. Each case corresponds to a trace, which is a finite sequence of events $\sigma \in (\mathcal{U}_E)^*$ such that each event appears only once, i.e., $\forall 1 \leqslant i < j \leqslant |\sigma| : \sigma_i \neq \sigma_j$.*

Table 2.1 shows two cases, i.e., "1" and "2". Consider for example case "1". Its corresponding trace is $\langle 35654423, 35654424, 35654425, 35654426, 35654427 \rangle$. In the remainder, a case can refer to its corresponding trace when the context is clear.

A meta event log directly consists of events and does not have a case notion to correlate events. For case-centric event data, we define a case-centric event log, which adds case information in a meta event log.

Definition 2.10 (Case-Centric Event Log) *A* case-centric event log *is a tuple $L = (E, act, attr, \preceq, Cas, tra)$, where*

- *$(E, act, attr, \preceq)$ is a meta event log,*
- *$case \in E \to \mathcal{U}_{Cas}$ maps events onto cases.*

[1] A total order is a binary relation that is (1) antisymmetric, i.e., $e_1 \preceq e_2$ and $e_2 \preceq e_1$ implies $e_1 = e_2$, (2) transitive, i.e., $e_1 \preceq e_2$ and $e_2 \preceq e_3$ implies $e_1 \preceq e_3$, and (3) total, i.e., $e_1 \preceq e_2$ or $e_2 \preceq e_1$.

Compared with a meta event log, a case-centric event log adds the case notion and maps each event to a case (identifier). The event data shown in Table 2.1 can be easily converted into a case-centric event log. An event log is a predefined structure for storing event data. The de facto standard for storing case-centric event logs on disk is the XES event log format [1, 157]. XES stands for eXtensible Event Stream and is the successor of the popular MXML event log format [152]. The XES standard stores information regarding the event log as a whole, the traces and the events belonging to the traces.

The XES log format introduced above is commonly used in existing process mining techniques. It forces users to pick a case notion when creating such logs, which limits its usage on artifact-centric systems. In this thesis, we will propose a novel log format (in Chapter 4), which follows the definition of meta event logs and discards the case notion, to organize the data from artifact-centric systems.

2.6 Summary

In this chapter, we presented necessary preliminaries for understanding this thesis. We started by introducing some basic mathematical notations, which serve as the basis for the definitions in this thesis. Then two mainstream data modeling languages were explained, which are referred to by the data perspective of our OCBC models. Since the data targeted by this thesis are from database tables, database systems were briefly discussed. Afterwards, a selected set of process modeling languages were presented. Among these languages, Petri nets and BPMN diagrams are widely used in academia and industry, respectively. Declare models are referred to by the behavioral perspective of OCBC models. The artifact-centric models are used to describe artifact-centric information systems. Since event logs are taken as input for process mining techniques, we introduced them at last.

Chapter 3

Object-Centric Event Data

As introduced in Chapter 1, this thesis proposes new process mining techniques to discover insights from data generated by artifact-centric information systems. Before presenting these novel techniques, the first step is to extract XOC logs from artifact-centric information systems, since process mining takes event logs as input.

This chapter and the next one in this thesis are focusing on the extraction part. Figure 3.1 zooms in the *extraction* part in the whole framework of this thesis and positions this chapter in the framework. More precisely, this chapter transforms data from the database of artifact-centric information systems into so-called *object-centric event data*. From the derived object-centric event data, Chapter 4 defines and extracts XOC logs. Note that these two chapters are closely connected, forming the extraction part.

In this chapter, we define the notion of *object-centric event data*, which contains the data perspective, behavioral perspective and the interactions between these two perspectives, as shown in Figure 3.2. More precisely, the data perspective is used to describe the data structure and data elements, the behavioral perspective is used to describe the events in the data, and the interactions are used to describe how events modify data elements. For instance, for a data set generated by an artifact-centric information system, the data perspective refers to database tables and the behavioral perspective refers to the operations on the system which generates transactions in database tables.

This chapter is organized as follows. In Section 3.1 we explain the data perspective. A data model is employed to describe the data structure (i.e.,

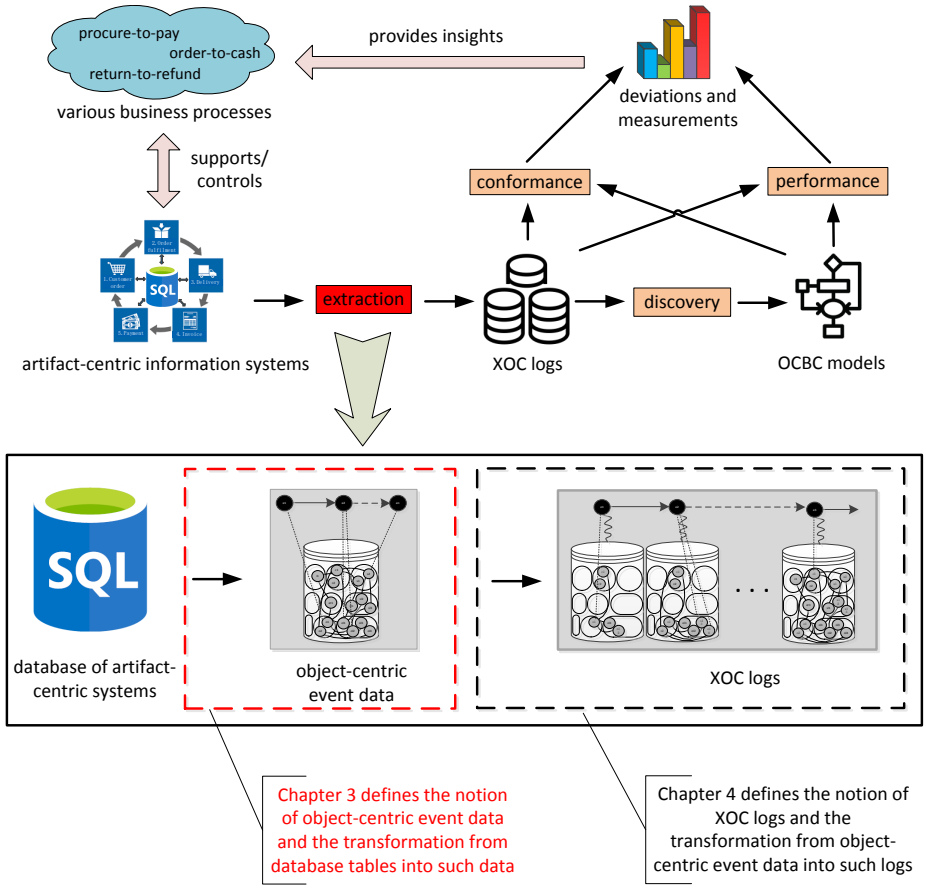


Figure 3.1: Positioning Chapter 3 in the whole framework.

database schema) and an object model is defined to represent data elements (i.e., database records). Section 3.2 illustrates the behavioral perspective, consisting of events extracted from various sources. Section 3.3 combines the data perspective and behavioral perspective, resulting in object-centric event data. In Section 3.4, we discuss the features of object-centric event data, and compare these with traditional case-centric data. Section 3.5 concludes this chapter.

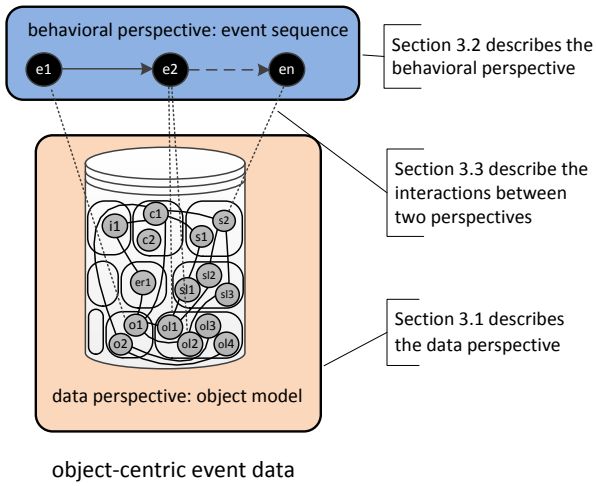


Figure 3.2: Object-centric event data consist of the data perspective, behavioral perspective and interactions between these two perspectives, which are illustrated in three sections in this chapter, respectively.

3.1 Data Perspective

In order to discover insights from a data set generated by artifact-centric systems, the first task is to analyze the data perspective of the data set. Here, the data perspective refers to the data structure and data elements. More precisely, in the context of databases, the data structure means the data schema and the data elements means the table records.

In this section, the data perspective is divided into three parts. First, we discuss the source to collect information related to the data perspective. Then the data structure is described by a data model. At last, data elements are abstracted as an object model.

3.1.1 Data Sources

Artifact-centric information systems are typically known as Enterprise Resource Planning (ERP) and/or Customer Relationship Management (CRM) systems, which support business functions related to sales, procurement, production,

accounting, etc. Some well-known examples are enterprise systems from vendors such as SAP (S/4HANA), Microsoft (Dynamics 365), Oracle (E-Business Suite), and Salesforce (CRM). Besides, there are also some free and open source alternatives such as Dolibarr and Odoo.[1]

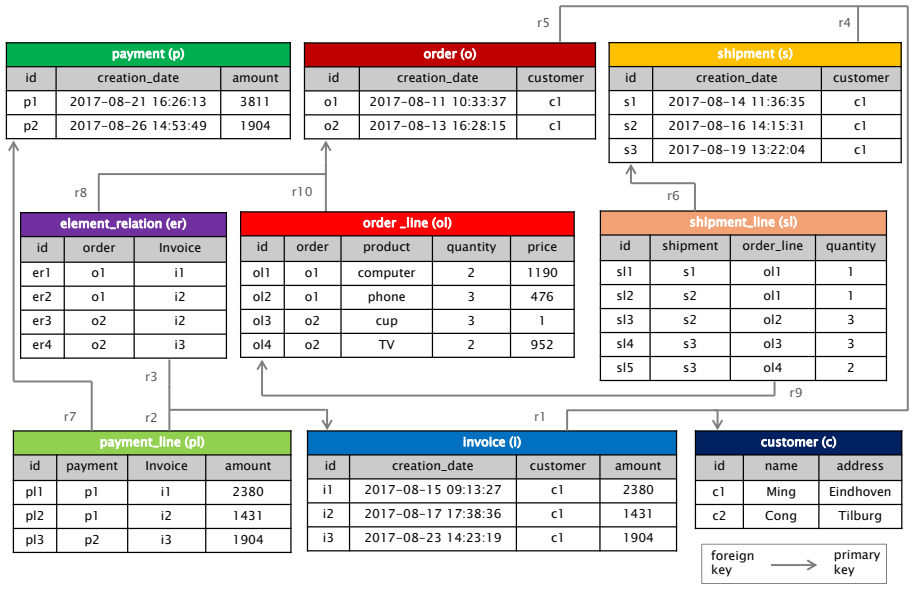


Figure 3.3: A fragment of database tables corresponding to the OTC scenario in an ERP system Dolibarr.

These systems provide interfaces for users to operate and all the transactions executed on the interfaces are stored in a relational database in an object-centric manner, i.e., transactions of the same category (e.g., orders) are recorded in the same table (e.g., the "order" table). The key feature of these systems is that they are built on top of database technology and may contain hundreds, if not thousands, of tables. For example, SAP has tens of thousands of tables with information about customers, orders, deliveries, etc. Also, Hospital Information Systems (HIS) and Product Lifecycle Management (PLM) systems have informa-

[1] Dolibarr ERP/CRM is an open source (webpage-based) software package for small and medium companies (www.dolibarr.org). It supports sales, orders, procurement, shipping, payments, contracts, project management, etc.

tion about many different entities scattered over a surprising number of database tables.

Figure 3.3 shows a fragment of database tables from a real ERP system, i.e., Dolibarr [89]. It contains nine tables, which store transactions in the OTC (order-to-cash) business scenario. More precisely, the "order" table contains all records related to customer orders, and each order has some order lines in the "order_line" table. The "shipment" and "shipment_line" tables contain information about the deliveries for orders. Each order can have some invoices in the "invoice" table, and the correspondences between them are indicated by the "element_relation" table. The "payment" and "payment_line" tables give the details of payments for invoices. Each order, shipment or invoice refers to a customer, whose information is shown in the "customer" table.

Besides, there are ten reference relations (i.e., $r1, r2, \ldots, r10$) between tables, which indicate the dependencies between records in different tables. For instance, $r10$ shows that the "order" column in the "order_line" table (a foreign key) references the "id" column in the "order" table (the primary key). It indicates that "order_line" records depend on "order" records. For instance, the first record in the "order_line" table depends on the first record in the "order" table.

The data set shown in Figure 3.3 provides information related to the data perspective of the business process. Next, we use this data set as an example to illustrate how to model the data structure and data elements, respectively.

3.1.2 Data Structure

The data structure describes how data are organized and stored in databases. It specifies the framework (backbone) of the database tables. For data modeling, UML class models [42] and Entity-Relationship (ER) models [25] are often used to describe the data schema of a database. Referring to them, in this part we formalize a data model to describe the database structure.

Figure 3.4 shows the idea to abstract the data structure, i.e., tables, columns of tables and dependencies between tables, as highlighted in the red dashed squares. More precisely, a table is abstracted as a *class*, columns are abstracted as *attributes* and the dependencies between tables are abstracted as relationships between classes. By integrating them, the data structure is abstracted as a *data model*.

Based on the idea, we next define classes, attributes and a function to indicate the correspondence between them. Note that each attribute can only have one type of values, e.g., the "creation_date" column in the "order" table in Figure 3.3

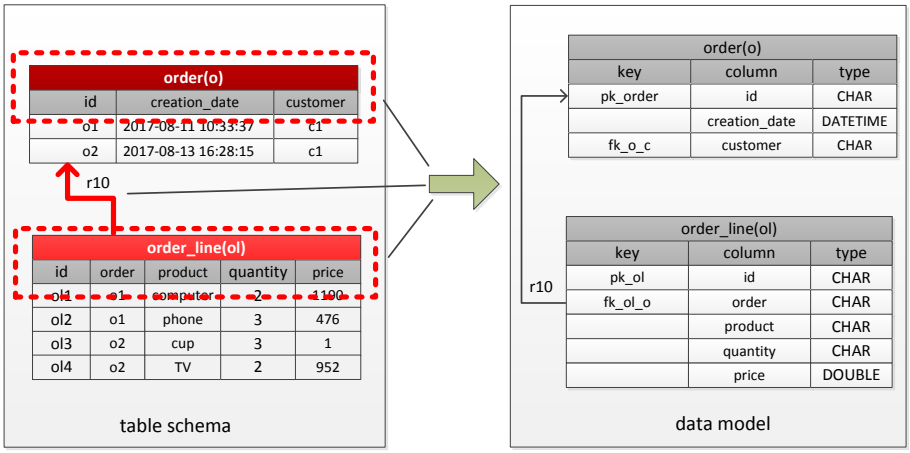


Figure 3.4: Abstracting the data structure (the highlighted parts, i.e., tables, columns and dependencies) as a data model.

can only have values of "DATETIME". Therefore, another function is defined to give the possible values for an attribute.

Definition 3.1 (Classes and Attributes) *Let $\mathcal{U}_C$ be the universe of* classes*, $\mathcal{U}_{Attr}$ be the universe of* attribute names*, and $\mathcal{U}_{Val}$ be the universe of* attribute values*. A table is represented by a class $c \in \mathcal{U}_C$, and a column in the table is represented by an attribute $attr \in \mathcal{U}_{Attr}$. Let $C \subseteq \mathcal{U}_C$ be a set of classes corresponding to all tables in a database and $Attr \subseteq \mathcal{U}_{Attr}$ be a set of attribute names corresponding to columns in all database tables.*

- *Function $classAttr \in C \rightarrow \mathcal{P}(Attr)$ gives all attribute names of a class where.*
- *Function $val \in Attr \rightarrow \mathcal{P}(V)$ maps each attribute name onto a set of values, where $V \subseteq \mathcal{U}_{Val}$ is a set of values.*

In the remainder, we use "attribute(s)" to refer to "attribute name(s)" when the context is clear.

According to Definition 3.1, a table is abstracted as a class and the columns of the table are abstracted as the attributes of the class, as shown in Figure 3.5. For instance, the table "tableA" (on the left) is abstracted as the class "classA" (on the right). Besides, the column "column1" is abstracted as the attribute "attr1". Function *classAttr* gives all attributes of a class, e.g., $classAttr(classA) =$

{*attr1*, *attr2*, *attr3*, *attr4*}. Note that in a table each column may have a type which restricts the possible values of the column. For instance, a timestamp column can only have times, i.e., values of the type "DATETIME". Accordingly, we define a function *val* which indicates the type of attribute values for each attribute, e.g., $val(attr3) = DATETIME$. Note that we focus on describing the data structure, rather than verifying if the values satisfy the data structure. In the remainder, we assume that the values always satisfy the requirements of the corresponding types.

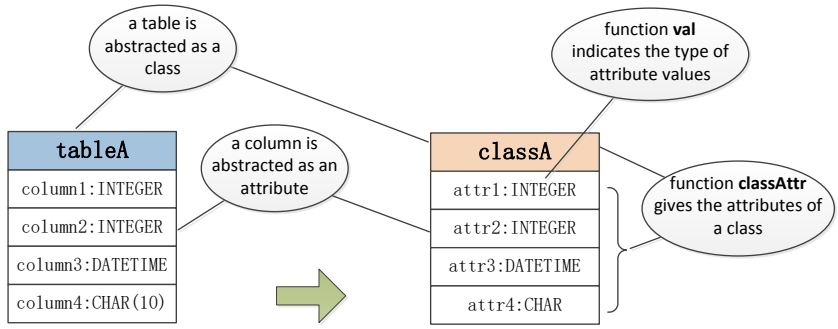


Figure 3.5: Abstracting tables and columns as classes and attributes, respectively.

In relational databases, each table must have precisely one primary key. A primary key corresponds to one or more columns whose values uniquely identify each record in the table. Besides, each table can have a set of foreign keys. A foreign key corresponds to one or more columns and refers to the primary key in another table. Foreign keys build the dependency relations between tables. Next, we define a generic format for both primary keys and foreign keys.

Definition 3.2 (Keys) *Let $\mathcal{U}_{KN}$ be the universe of* key names. *A key $k = (kn, Attr) \in \mathcal{U}_{KN} \times \mathcal{P}(\mathcal{U}_{Attr})$ is a tuple of a key name and a set of attributes. Let $\mathcal{U}_K$ be the universe of keys, where any two keys cannot have the same key name, i.e., $\forall (kn, Attr), (kn', Attr') \in \mathcal{U}_K : kn = kn' \Rightarrow (kn, Attr) = (kn', Attr')$.*

Function $keyAttr \in K \rightarrow \mathcal{P}(Attr)$ gives the set of attributes of a key where $K \supseteq \mathcal{U}_K$ is a set of keys and $Attr \supseteq \mathcal{U}_{Attr}$ is a set of attributes corresponding to the keys.

In the context of databases, a key is defined as a constraint spanning one or more columns with a unique constraint name using an SQL sentence. Accordingly, a key is specified as a combination of a unique key name and a set of attributes

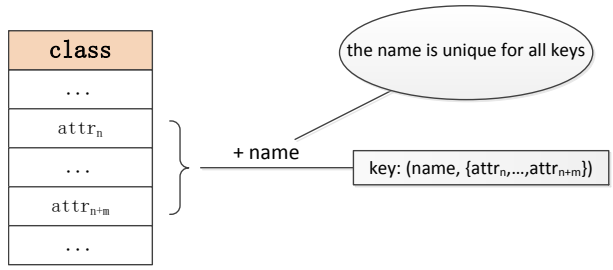


Figure 3.6: A key is a combination of a unique name and a set of attributes.

as shown in Figure 3.6. Consider for example the "order" table in Figure 3.3. Its primary key is $pk = (pk\_order, \{id\})$ where *pk_order* is the key name and *id* is an attribute. Besides, $(fk\_order, \{customer\})$ is one of its foreign keys, where *fk_order* is the key name and *customer* is an attribute. Note that it is possible that two keys have the same set of attributes, but two keys can never have the same key name. For instance, the primary keys of the "order" table and "shipment" table can have the same set of attributes, e.g., $\{id\}$, but they should be given different key names, e.g., *pk_order* for "order" and *pk_shipment* for "shipment". Function *keyAttr* gives the attributes of a key, e.g., $keyAttr(pk) = \{id\}$. Note that the input of *keyAttr* is a key (e.g., *pk*) rather than a key name (e.g., *pk_order*).

A foreign key must reference precisely one primary key to form a dependency relation between two tables.[2] Accordingly, each attribute of the foreign key must reference one attribute of the primary key. In the following, we define two functions to map foreign keys to primary keys on the key level and attribute level, respectively.

Definition 3.3 (Mapping Foreign Keys to Primary Keys) *Let $PK \subseteq \mathcal{U}_K$ be a set of primary keys and $FK \subseteq \mathcal{U}_K$ be a set of foreign keys, where PK and FK are disjoint sets (i.e., $PK \cap FK = \emptyset$).*

- *Function $keyRel \in FK \rightarrow PK$ maps each foreign key onto a primary key, i.e., specifies FK-PK reference relations.*
- *Function $refAttr \in FK \times Attr \not\rightarrow Attr$ maps each pair of a foreign key and an attribute (of the foreign key) onto an attribute of the corresponding primary*

[2] One foreign key $fk_1 = (kn_1, Attr_1)$ can only reference one primary key. It is possible that another foreign key $fk_2 = (kn_2, Attr_2)$ (referencing another primary key) uses the same attributes as fk_1, and a different key name, i.e., $Attr_1 = Attr_2$ and $kn_1 \neq kn_2$. Note that this situation does not mean that one foreign key can reference two primary keys.

key, where $Attr \subseteq \mathcal{U}_{Attr}$ is a set of attributes.

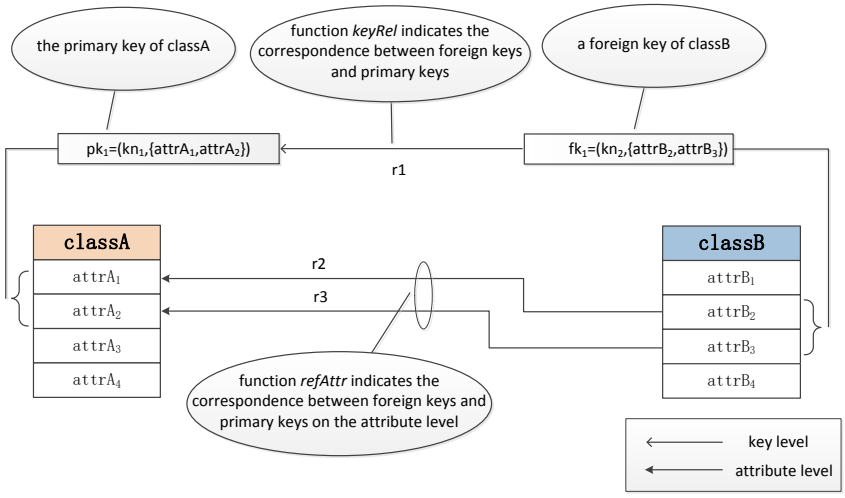


Figure 3.7: Mapping foreign keys onto primary keys on the key and attribute levels.

For a foreign key, function *keyRel* gives its corresponding primary key and function *refAttr* shows the correspondence on the attribute level. Consider for example the two classes "classA" and "classB" in Figure 3.7. The foreign key of "classB" fk_1 references the primary key of "classA" (indicated by *r1*), i.e., $keyRel(fk_1) = pk_1$. On the attribute level, $attrB_2$ references $attrA_1$ (indicated by *r2*) and $attrB_3$ references $attrA_2$ (indicated by *r3*), i.e., $refAttr(fk_1, attrB_2) = attrA_1$ and $refAttr(fk_1, attrB_3) = attrA_2$.

Note that each attribute of the foreign key can only reference an attribute of its corresponding primary key (i.e., it cannot reference one of the other attributes). In other words, for any $fk \in FK$, $\forall attr \in keyAttr(fk) : refAttr(fk, attr) \in keyAttr(pk)$, where pk is the primary key referenced by fk (i.e., $pk = keyRel(fk)$). For instance, the attribute $attrB_2$ of the foreign key fk_1 cannot reference $attrA_3$, as $attrA_3$ is not an attribute of the primary key pk_1 in Figure 3.7.

Besides, it is impossible that two attributes of a foreign key reference the same attribute of the corresponding primary key. That is, for any $fk \in FK$, $\forall attr, attr' \in keyAttr(fk) : refAttr(fk, attr) = refAttr(fk, attr') \implies attr = attr'$. For instance, the attributes $attrB_2$ and $attrB_3$ of the foreign key fk_1 cannot reference the same

attribute $attrA_1$ of the primary key pk_1 in Figure 3.7.

Up to now, we have abstracted tables as classes and dependencies between tables as references between foreign keys and primary keys. Now, we formalize the data model, which integrates all the notions explained above, to represent the data structure.

Definition 3.4 (Data Model) *A data model is a tuple* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *such that:*

- $C \subseteq \mathcal{U}_C$ *is a set of classes,*
- $Attr \subseteq \mathcal{U}_{Attr}$ *is a set of attribute names,*
- $classAttr \in C \to \mathcal{P}(Attr)$ *is a function mapping each class onto a set of attribute names,*
- $val \in Attr \to \mathcal{P}(V)$ *is a function mapping each attribute onto a set of values,*
- $K \subseteq \mathcal{U}_{KN} \times \mathcal{P}(Attr)$ *is a set of keys. PK is a set of primary keys and FK is a set of foreign keys, where PK and FK are disjoint sets (i.e.,* $PK \cap FK = \emptyset$*) and* $K = PK \cup FK$*,*
- $classPK \in C \to PK$ *is a total injective function mapping each class onto a primary key,*
- $classFK \in C \to \mathcal{P}(FK)$ *is a total injective function mapping each class onto a set of foreign keys, such that for any* $c_1, c_2 \in C : classFK(c_1) \cap classFK(c_2) = \emptyset$*,*
- $keyRel \in FK \to PK$ *is a function mapping each foreign key onto a primary key,*
- $keyAttr \in K \to \mathcal{P}(Attr)$ *is a function mapping each key onto a set of attributes, such that* $\forall k \in K : \exists c \in C : (k \in \{classPK(c)\} \cup classFK(c) \wedge keyAttr(k) \subseteq classAttr(c))$*, and*
- $refAttr \in FK \times Attr \not\to Attr$ *is a function mapping each pair of a foreign key and an attribute onto an attribute of the corresponding primary key, such that*
  - $\forall fk \in FK : \{a \mid (fk, a) \in dom(refAttr)\} = keyAttr(fk) \wedge \{a' \mid \exists (fk, a) \in dom(refAttr) : refAttr(fk, a) = a'\} = keyAttr(keyRel(fk))$*, and*
  - $\forall (fk, a) \in dom(refAttr) : val(refAttr(fk, a)) = val(a)$.

$\mathcal{U}_{DM}$ *is the universe of data models.*

In the context of databases, a *data model* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ describes the structure of a database (or a part of a database involved in a particular analysis). More precisely, a class $c \in C$ represents a table (i.e., C represents all tables in the database) while an attribute $attr \in Attr$ represents a column in a table (i.e., $Attr$ represents all table columns). Function $classAttr$ specifies the attributes of a class and function val indicates possible values for an attribute.

PK and FK represent all primary keys and foreign keys in the database, respectively. PK and FK are disjoint sets as it is impossible that a key serves as

both a primary key and a foreign key.[3] Functions *classPK* and *classFK* define the primary key and foreign keys for each class, respectively. Note that different classes cannot have the same primary key or foreign key (indicated by the total injective function and $c_1, c_2 \in C : classFK(c_1) \cap classFK(c_2) = \emptyset$).

For each key, function *keyAttr* specifies all its corresponding attributes. Different keys can have the same attributes. The attributes of a key should be contained by the attributes of the class corresponding to the key (i.e., $\forall k \in K : \exists c \in C : (k \in \{classPK(c)\} \cup classFK(c) \wedge keyAttr(k) \subseteq classAttr(c))$).

For a foreign key, *keyRel* indicates its corresponding primary key, and *refAttr* specifies the mapping relations between the attributes of the foreign key and the attributes of the primary key. Note that each attribute of a foreign key should be linked to an attribute of is corresponding primary key and vice versa, required by *refAttr*. In other words, for a foreign key, i.e., $fk \in FK$, (i) the possible attributes involved in the domain of *refAttr* ($\{a \mid (fk, a) \in dom(refAttr)\}$) are all attributes of the foreign key ($\{keyAttr(fk)\}$); (ii) the possible attributes returned by *refAttr* ($\{a' \mid \exists (fk, a) \in dom(refAttr) : refAttr(fk, a) = a'\}$) are all attributes of the primary key corresponding to the foreign key ($keyAttr(keyRel(fk))$). Besides, the two linked attributes should have the same type of values, i.e., $\forall (fk, a) \in dom(refAttr) : val(refAttr(fk, a)) = val(a)$.

For instance, the database in Figure 3.3 can be represented by a data model where $C = \{p, o, s, er, ol, sl, pl, i, c\}$ and $Attr = \{id, creation\_date, \ldots, address\}$. Based on the dependencies between tables (i.e., *r1*, *r2*,..., *r10*), we assign a name (e.g., *pk_order*) to each primary key or foreign key, resulting in $PK = \{(pk\_order, \{id\}), (pk\_shipment, \{id\}), \ldots, (pk\_customer, \{id\})\}$ and $FK = \{(fk\_order, \{customer\}), (fk\_shipment, \{customer\}), \ldots, (fk\_invoice, \{customer\})\}$. Consider for example the class *o* (i.e., "order") to understand the functions related to classes. The set of its attributes is $classAttr(o) = \{id, creation\_date, customer\}$, its primary key is $classPK(o) = (pk\_order, \{id\})$ and it has only one foreign key, i.e., $classFK(o) = \{(fk\_order, \{customer\})\}$. For an example attribute "creation_date", the type of its possible values is $val(creation\_date) = DATETIME$. The foreign key $fk = (fk\_order, \{customer\})$ is used to explain the functions related to foreign keys. Its corresponding primary key is $keyRel(fk) = (pk\_customer, \{id\})$ and the set of its attributes is $keyAttr(fk) = \{customer\}$. Besides, its attribute *customer* references the attribute *id*, i.e., $refAttr(fk, customer) = id$.

Figure 3.8 presents a visualization solution for the data model explained above. The model is depicted as a graph consisting of nodes (consisting of

[3] A primary key and a foreign key may have the same set of attributes. However, they are not the same key as they have different key names.

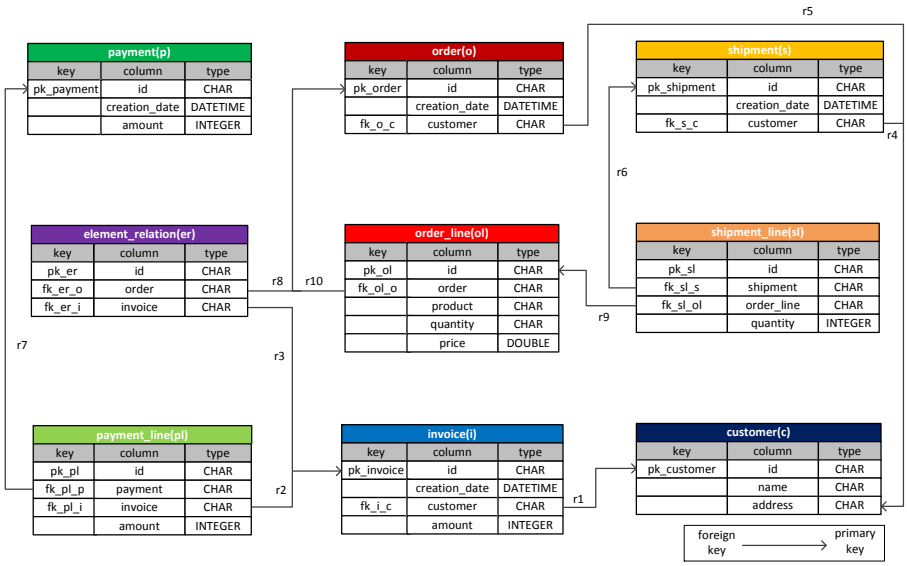


Figure 3.8: The data model describing the structure of the database in Figure 3.3.

smaller grids in rows and columns) and edges between nodes. More precisely, nodes represent classes and edges represent references between foreign keys and primary keys in the data model. For instance, the node on the top left corresponds to the class “payment”. The first row colored in green in the node shows the name of the class, i.e., “payment(p)” (the letter in the brackets shows the abbreviation of the name). The next rows are divided into three columns, i.e., “key”, “column” and “type”, which show the key names, attribute names and value types, respectively. Note that each primary key name has a prefix of “pk” and each foreign key name has a prefix of “fk”. For instance, “pk_order” is the primary key and “fk_o_c” is one foreign key of the “order” table. Each edge represents a reference relation, which has an arrow from the foreign key (in the child table) to the primary key (in the parent table). The labels on the edges are used to refer to the relations between foreign keys and primary keys. For instance, $r7$ refers to the relation between the foreign key “fk_pl_p” in “payment_line” and the primary key “pk_payment” in “payment”.

3.1.3 Data Elements

In Section 3.1.2, the data structure is abstracted as a data model. In this part, we shift attention to data elements, that can be considered as the instances allowed by the data structure and provide information of the real executions of business processes.

Figure 3.9 shows the idea to abstract data elements. More precisely, an individual record in one of the database tables is abstracted as an object, and dependency relations among records are abstracted as object relations. By integrating them, the data elements in the database (the highlighted parts in the red dashed squares) are abstracted as an object model.

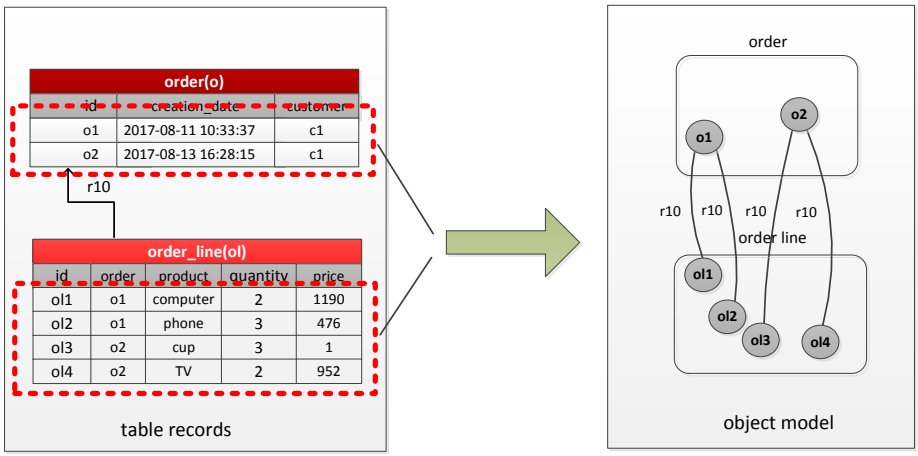


Figure 3.9: Abstracting data elements (the highlighted parts, i.e., table records) as an object model.

In databases, a record is an instance of a database table, which instantiates all columns of the table. Accordingly, in the context of a data model, an object (representing the record) is an instance of a class (representing the table), which assigns values to all attributes of the class. Therefore, an object should have a corresponding class and a function (assigning values to attributes), to contain all information of a record.

Definition 3.5 (Objects) *Let* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model and* $M^{DM} = \{map \in Attr \nrightarrow V \mid \forall attr \in$

$dom(map) : map(attr) \in val(attr)\}$ *be the set of mappings.* $O^{DM} = \{(c, map) \in C \times M^{DM} \mid dom(map) = classAttr(c) \wedge \forall attr \in keyAttr(classPK(c)) : map(attr) \neq NULL\}$ *is the set of all possible objects of DM.*

Based on a data model, Definition 3.5 gives the formalization of objects. An object consists of a class c and a function *map*, where c shows the class of the object and *map* instantiates all attributes of the class. As shown in Figure 3.10, the first record in the "order" table can be formalized as an object $o = (c, map)$ where $c = order$, $dom(map) = \{id, creation\_date, customer\}$, and $map(id) = o1$, $map(creation\_date)$ =2017-08-11 10:33:37 and $map(customer) = c1$.

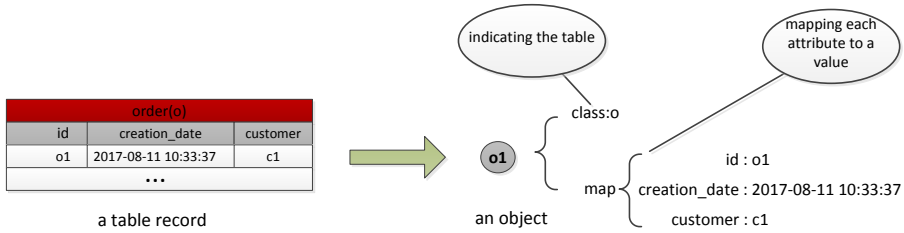


Figure 3.10: Abstracting a table record as an object.

Note that it is possible that a record is added to a table only assigning values to primary key columns (i.e., the columns corresponding to the primary key). In other words, attributes of the primary key must be given values (i.e., $attr \in keyAttr(classPK(c)) : map(attr) \neq NULL$) and the other attributes may be empty (or usually instantiated as the value "NULL" by default).

In databases, a record can reference another record through a reference (i.e., FK-PK) relation. Accordingly, an object can be correlated to another object through a so-called *object relation*. Note that one class may have multiple foreign keys, such that objects can be correlated through different FK-PK relations. For instance, the first record *er1* in the "element_relation" table in Figure 3.3 (i) references the record *o1* in the "order" table through *r8* and (ii) references the record *i1* in the "invoice" table through *r3*. Therefore an object relation should have a corresponding type, indicating through which FK-PK relation two objects are correlated.

Definition 3.6 (Class Relationships (Object Relation Types)) *Let* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model. A*

class relationship is a tuple $r = (fk, pk) \in FK \times PK$ *where* $keyRel(fk) = pk$. $R^{DM} = \{(fk, pk) \in FK \times PK \mid keyRel(fk) = pk\}$ *is the set of all possible class relationships of DM.*

Class relationships correspond to reference (FK-PK) relations between tables. If a foreign key *fk* of class *c* (i.e., $fk \in classFK(c)$) references a primary key *pk* of a class c' (i.e., $pk = classPK(c')$), we say that there exists a class relationship from *c* to c'. For instance, in Figure 3.8 there exists a class relationship from class "order line" to class "order", as the foreign key *fk_ol_o* of table "order_line" references the primary key *pk_order* of table "order" (indicated by *r10*).

By using class relationships to indicate types, an object relation is defined as a tuple of a class relationship, a source object and a target object, which means that the source object references the target object through the class relationship.

Definition 3.7 (Object Relations) *Let* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model. There exists an object relation* $rel = (r, o, o') \in R^{DM} \times O^{DM} \times O^{DM}$ *of class relationship* $r = (fk, pk)$ *from object* $o = (c, map)$ *to object* $o' = (c', map')$*, denoted as* $o \overset{r}{\mapsto} o'$*, if*

- $fk \in classFK(c)$,
- $pk = classPK(c')$, *and*
- $\forall attr \in keyAttr(fk) : map(attr) = map'(refAttr(fk, attr))$.

Rel^{DM} *is the set of all possible object relations of DM, i.e.,* $Rel^{DM} = \{(r, o, o') \in R^{DM} \times O^{DM} \times O^{DM} \mid o \overset{r}{\mapsto} o'\}$.

There exists an object relation of type $r = (fk, pk)$ between an object $o = (c, map)$ and another object $o' = (c', map')$, if (i) *fk* is one of the foreign keys of *c*, (ii) *pk* is the primary key of c' and (iii) each attribute value (corresponding to the foreign key *fk*) of *o* is equal to the corresponding attribute value of o', i.e., $\forall attr \in keyAttr(fk) : map(attr) = map'(refAttr(fk, attr))$.

Consider the objects in Figure 3.11 to understand how to identify the object relations. Assume that object $ol1 = (c, map)$ corresponds to the first record in the "order_line" table and object $o1 = (c', map')$ corresponds to the first record in the "order" table. As shown in the data model in Figure 3.8, there exists a class relationship *r10* connecting the foreign key $fk = (fk\_ol\_o, \{order\})$ of the "order_line" table to the primary key $pk = (pk\_order, \{id\})$ of the "order" table. In other words, $r10 = (fk, pk)$ and $refAttr(fk, order) = id$. For the object *ol1*, the value in its attribute "order" is *o1*, i.e., $map(order) = o1$, which is equal to the value in the attribute "id" of *o1*, i.e., $map'(id) = o1$. Therefore, there exists an object relation $(r10, ol1, o1)$ between *ol1* and *o1*, i.e., $ol1 \overset{r10}{\mapsto} o1$. Similarly, we can also derive $ol2 \overset{r10}{\mapsto} o1$

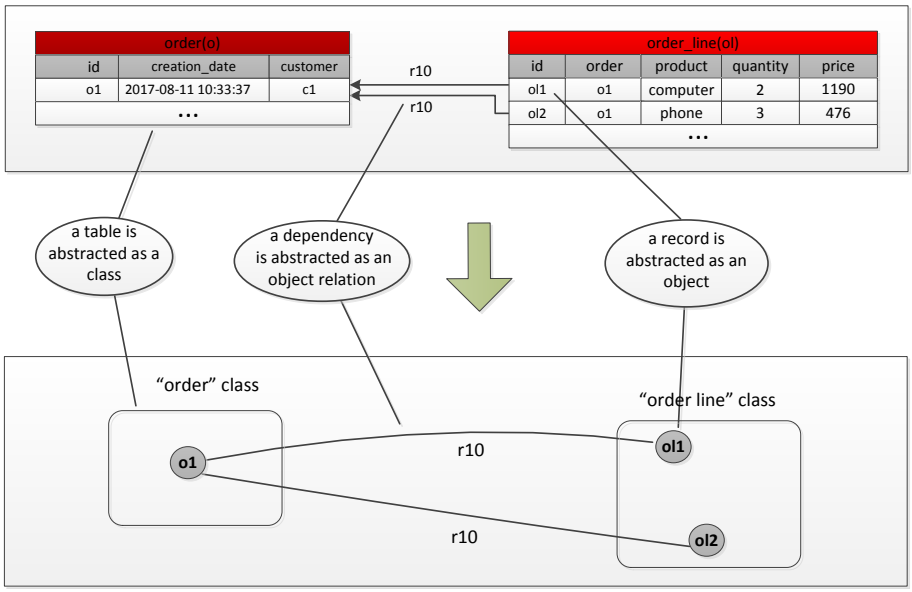


Figure 3.11: Abstracting dependencies between records as object relations between objects.

Definition 3.8 (Extracting Object Relations) *Let DM be a data model and $Obj \subseteq O^{DM}$ be a set of objects. $extractRel \in \mathcal{P}(O^{DM}) \rightarrow \mathcal{P}(Rel^{DM})$ is a function extracting all possible object relations from a set of objects such that $extractRel(Obj) = \{(r, o, o') \in R^{DM} \times Obj \times Obj \mid o \overset{r}{\mapsto} o'\}$.*

Definition 3.8 presents a function which extracts all possible object relations from a given set of objects. Consider for example the set of objects $Obj = \{o1, ol1, ol2\}$ in Figure 3.11. $extractRel(Obj) = \{(r10, ol1, o1), (r10, ol2, o1)\}$, as $ol1 \overset{r10}{\mapsto} o1$ and $ol2 \overset{r10}{\mapsto} o1$. Specifically, with all possible objects of DM as input, the function returns all possible object relations of DM, i.e., $Rel^{DM} = extractRel(O^{DM})$.

Up to now, we have explained objects and object relations to abstract the records and dependencies between records in database tables. Next, we integrate them to form a so-called *object model*, which describes the whole set of elements of a database.

Definition 3.9 (Object Model) *Let* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model. An object model of DM is a tuple* $OM = (Obj, Rel, class, objectAttr)$*, where*

- $Obj \subseteq O^{DM}$ *is a set of objects,*
- $Rel \subseteq R^{DM} \times Obj \times Obj$ *is a set of object relations,*
- $class \in Obj \rightarrow C$ *maps objects onto classes, such that* $\forall o = (c, map) \in Obj : class(o) = c$*, and*
- $objectAttr \in Obj \rightarrow (\mathcal{U}_{Attr} \nrightarrow \mathcal{U}_{Val})$ *maps objects onto a partial function assigning values to some attributes, such that* $\forall o = (c, map) \in Obj : objectAttr(o) = map$*.*

$\mathcal{U}_{OM}$ *is the universe of object models.*

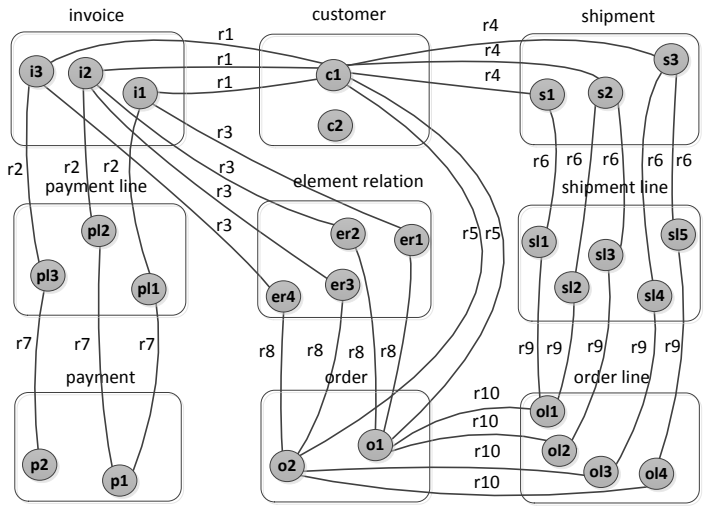


Figure 3.12: An object model representing all data elements of the database in Figure 3.3.

An object model consists of a set of objects and a set of object relations between these objects. Functions *class* and *objectAttr* are defined to easily refer to the classes and attributes of objects. Actually, they are contained implicitly in the objects. In the database context, an object model represents a snapshot (i.e., a state) of the database at some point in time, i.e., objects represent records while relations represent dependencies between records. All data elements in

Figure 3.3 can be abstracted as the object model $OM = (Obj, Rel, class, objectAttr)$ in Figure 3.12. Obj consists of twenty-eight objects, e.g., $o1$ and $ol1$, which correspond to the first record in the "order" and "order line" tables respectively. Rel consists of thirty-six object relations, e.g., $(r10, ol1, o1)$, which indicates that the record $ol1$ references the record $o1$ through $r10$. Consider the object $o1$ to understand $class$ and $objectAttr$. $class(o1) = order$ and $objectAttr(o1) = map$ where $map(id) = o1$, $map(creation\_date) = 2017-08-11\ 10:33:37$ and $map(customer) = c1$.

In databases, records must have unique primary key values (i.e., the combinations of values in the columns corresponding to the primary key are unique). For instance, the two records in the "order" table in Figure 3.3 cannot have the same value in the "id" column. Besides, if the foreign key values (i.e., values in the columns corresponding to a foreign key) of a record are not "NULL" (i.e., not empty), the referenced record must exist. For instance, in the "order_line" table, the first record $ol1$ has the value $o1$ in the "order" column (corresponding to a foreign key referencing the primary key of "order" table), which means that $ol1$ references another record $o1$. Therefore, $o1$ must exist in the "order" table. Note that the foreign key values of a record may be "NULL" (i.e., empty) at some moment, which means that the record does not yet reference a potential record.

Given a set of tables, if all records satisfy the rules explained above, the set of tables is a possible state of the database, i.e., the records are valid for the database structure. As a data model represents the structure of the database and an object model represents the state of the database, we can use the object model to check its consistency. More precisely, we can check if an object model is valid for a data model, i.e., the state represented by the object model is a possible state of the database represented by the data model.

Definition 3.10 (Valid Object Model) *Let $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ be a data model and $OM = (Obj, Rel, class, objectAttr)$ be an object model. OM is valid for DM if and only if:*

- $\forall (c, map), (c, map') \in Obj : (\forall attr \in keyAttr(classPK(c)) : map(attr) = map'(attr)) \Rightarrow map = map'$*, i.e., primary key values must be unique,*
- $\forall (c, map) \in Obj : \forall fk \in classFK(c)$:
  - $\forall attr \in keyAttr(fk) : map(attr) = NULL$*, i.e., the foreign key values are not instantiated, or*
  - $\exists (c', map') \in Obj : keyRel(fk) = classPK(c') \wedge \forall attr \in keyAttr(fk) : map(attr) = map'(refAttr(fk, attr))$*, i.e., the referenced object must exist,*
- $Rel = extractRel(Obj)$*, i.e., all possible object relations are contained by Rel.*

An object model OM is valid if (i) each object in Obj is unique (i.e., its

primary key values are unique), (ii) all objects referenced by other objects exist in *Obj*, (iii) *Rel* is complete (i.e., it contains any object relation between each two objects, in which one references the other) and not redundant (i.e., each object relation contained by it corresponds to a reference between two objects). For instance, the object model in Figure 3.12 is valid for the data model in Figure 3.8 as it satisfies all the rules in Definition 3.10. It is not valid if we (i) add an object which has the same primary key values as any already existing object, (ii) remove a referenced object, e.g., $o1$ (referenced by $ol1$ and $ol2$), or (iii) remove any object relation (e.g., $(r10, ol1, o1)$) or add an inexistent object relation (e.g., $(r10, sl1, o1)$).

3.2 Behavioral Perspective

Section 3.1 introduced the data perspective of object-centric event data, i.e., data structure (the schema of a database) and data elements (records of a database). Essentially, object-centric event data are generated by business transactions in information systems. In other words, transactions in these systems modify database tables by adding, updating or deleting records, which leave time-related footprints in various sources. The footprints can be extracted as events, which form the behavioral perspective and provide a business process view to show how the transactions are executed.

In this section, we illustrate the behavioral perspective of object-centric event data in three parts. The first part discusses various event sources where events can be extracted. Then we explain the events at the database level, i.e., database changes, which indicate the changes of records in the database. At last, the events on the information system level are extracted, which have more intuitive meanings and are easier to understand for users.

3.2.1 Event Sources

Information systems (such as WFM/BPM systems, ERP/CRM systems, rule-based systems, call center software, high-end middleware, etc.) provide a wealth of event data, which can be explored to discover insights to reflect executed business processes. Events may be "machine events" (e.g., an X-ray machine, an ATM, or a baggage handling system), "organization events" (e.g., the analysis of a blood sample in a hospital or a delivery for a customer order in a sales company), "web events" on the Internet (e.g., web services) and "life events" on the social media (e.g., messages on Facebook, twitter and LinkedIn).

The term Internet of Events (IoE) was proposed in [134] to refer to all events from various sources as shown in Figure 3.13. Typically, these sources are composed of: (i) the *Internet of Content* (IoC), e.g., traditional web pages, articles, encyclopedia like Wikipedia, YouTube, e-books and newsfeeds; (ii) the *Internet of People*, e.g., interactions of people, such as e-mail, facebook, twitter, forums, LinkedIn; (iii) the Internet of Things (IoT), e.g., things may have an internet connection or tagged using Radio-Frequency Identification (RFID), Near Field Communication (NFC); (iv) the Internet of Locations (IoL), e.g., data that have a spatial dimension (geospatial attributes).

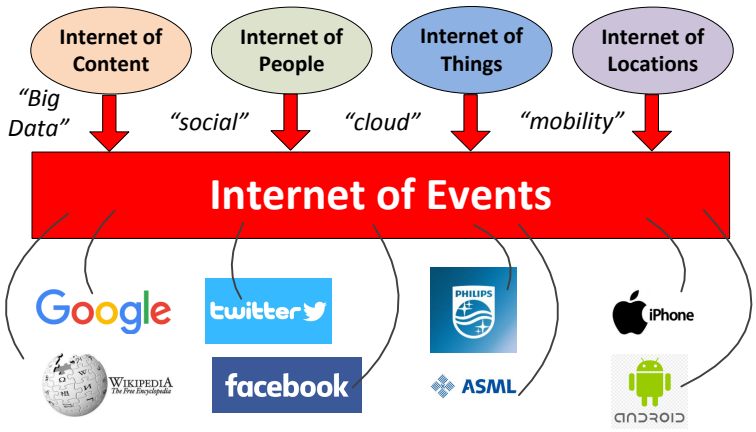


Figure 3.13: Internet of Events (IoE): event data are generated from a variety of sources.

Process mining aims to reveal meaningful insights from event data, e.g., discover process models, identify bottlenecks, detect violations and anticipate problems. In this thesis, we mainly focus on event data from artifact-centric information systems. Note that such event data are only a small part of IoE. In future, it is possible to exploit the possibility of applying our techniques to data from other domains, such as social media.

All the transactions executed on artifact-centric information systems are stored in relational database tables, in which events are recorded often in an implicit and scattered manner. Therefore, Database Management Systems (DBMSs) can be exploited to extract events, as explained next.

Redo Logs

In practice, there exist various DBMSs, in which some widely used examples are Oracle RDBMS (Oracle), SQL server (Microsoft), PostgreSQL (PostgreSQL Global Development Group), DB2 (IBM) and Sybase (SAP). For data integrity and recovery reasons, all of these systems have facilities to record database changes [135].

For example, in the Oracle RDBMS environment, *redo logs* record the history of database changes, which can be queried by Oracle LogMiner, a utility provided by Oracle [135]. Besides, every Microsoft SQL Server database has a transaction log that records all database modifications. Sybase IQ also provides a transaction log. In this thesis, we use redo logs to refer to such logs.

Index	Time	Redo
1	2017-08-11 10:33:37	insert into “order” (“id”,“creation_date”,“customer”) values (“o1”,“2017-08-11 10:33:37”,“c1”)
2	2017-08-11 10:33:37	insert into “order_line” (“id”,“order”,“product”,“quantity”,“price”) values (“ol1”,“o1”,“computer”,“2”,“1190”)
3	2017-08-11 10:33:37	insert into “order_line” (“id”,“order”,“product”,“quantity”,“price”) values (“ol2”,“o1”,“phone”,“3”,“476”)
...	...	...

Table 3.1: A fragment of a redo log in which each line corresponds to a change in databases.

Table 3.1 shows a segment of a redo log [49]. Each row in the redo log corresponds to one execution of an SQL sentence in the database. For instance, the first row records the execution of an insertion sentence which inserts a record into the “order” table. This source of data has the potential to create a full history of what has happened on databases [51]. In other words, it can be used to extract events, which modify some objects (records) in tables. For instance, the first row may correspond to one event that happened at “2017-08-11 10:33:37”, which changes the object $o1$.

Change Tables

Some SAP systems provide change tables (i.e., CDHDR and CDPOS) to record database changes as shown in Table 3.2 and Table 3.3, which can be used to detect events [115]. CDHDR is the change head table which records all modifications in the database. In the CDHDR table, the “CHANGENR” column can be considered as the primary key, i.e., each change performed on the database

OBJECTCLAS	CHANGENR	OBJECTID	USERNAME	UDATE	UTIME	TCODE	CHANGE_IND
o	0001	0000001	USER1	2017-08-11	10:33:37	0001	I
ol	0002	0000002	USER1	2017-08-11	10:33:37	0001	I
ol	0003	0000003	USER1	2017-08-11	10:33:37	0001	I
o	0004	0000004	USER1	2017-08-13	16:28:15	0001	I
...	...	...	...	...	...	...	...

Table 3.2: A fragment of the CDHDR table.

results in an entry with a unique change id in the column. Besides, the "UDATE" and "UTIME" columns record the time of changes and "CHANGE_IND" indicates the type of changes, e.g., "I" means that the change inserts a record in the database.

OBJECTCLAS	CHANGENR	TABNAME	TABKEY	FNAME	VALUE_NEW	VALUE_OLD
o	0001	order	o1	id	o1	-
o	0001	order	o1	creation_date	2017-08-11 10:33:37	-
o	0001	order	o1	customer	c1	-
ol	0002	order_line	ol1	id	ol1	-
ol	0002	order_line	ol1	order	o1	-
ol	0002	order_line	ol1	product	computer	-
ol	0002	order_line	ol1	quantity	2	-
ol	0002	order_line	ol1	price	1190	-
...	...	...	...	...	...	...

Table 3.3: A fragment of the CDPOS table.

In contrast, the CDPOS table is the item table for changes, which records the details of each modification, i.e., what values have been changed exactly. It also has the "CHANGENR" column which references the "CHANGENR" column in the CDHDR table. The "TABNAME" column and "FNAME" column indicate the table and field that are impacted by the change, respectively. The "TABKEY" column gives the primary key value of the impacted record. The "VALUE_OLD" column and "VALUE_NEW" column give the values for the impacted field before the change and after the change, respectively.

Based on the CDHDR and CDPOS change tables in SAP, we can reconstruct the full history of the database. Note that one change in the CDHDR table may correspond to multiple records in the CDPOS table. For instance, the change "0001" in the CDHDR table corresponds to the first three rows in the CDPOS table, which indicate the insertion of the record $o1$ in the "order" table in Figure 3.3.

Timestamp Columns

In addition to redo logs and change tables, the timestamp columns in tables (e.g., "creation_date" column in "order" table in Figure 3.3) can also be used to extract events with the support of domain knowledge.

For instance, assume that we can get from the domain knowledge that each "create order" event adds one record in the "order" table and one or more records in the "order line" table. Then a "create order" event that happened at "2017-08-11 10:33:37" can be extracted from the first row in the "order" table in Figure 3.3, which inserts three records $o1$, $ol1$ and $ol2$.

In summary, the events extracted from various data sources introduced in this section can be divided into two levels, the database level and information system level. On the database level events refer to the database changes, and on the information system level events refer to the operations of users on interfaces. We explain them in the following sections.

3.2.2 Events at the Database Level

In this section, we analyze events at the database level, i.e., database changes directly extracted from files such as redo logs and change tables. More precisely, the analysis does not focus on a specific source of a particular DBMS. We try to define some basic and generic definitions to formalize the events from different sources.

As explained in Section 3.2.1, redo logs can monitor changes in the database. In particular, we assume that we can see when a record is inserted, updated, or deleted. Figure 3.14 shows three typical database changes. The top one indicates that one record is inserted into the "order" table while the bottom one shows the opposite situation, i.e., one "order" record is deleted. The middle one means that the value of the "quantity" attribute in the record $ol1$ is updated from "2" to "4". Clearly, changes are different in three perspectives:

- *Operation types*. Changes may correspond to different types of SQL sentences, i.e., *insertion*, *update* and *deletion*. For instance, the three changes correspond to three different types of sentences.
- *Classes*. Changes may impact records in different tables, i.e., impact objects of different classes. For instance, the top change impacts the "order" table while the middle change impacts the "order_line" table.
- *Attributes*. Changes may impact different attributes of records. For instance, the middle change in Figure 3.14 only updates the value of the "quantity" attribute. It is possible that another change updates the values of other

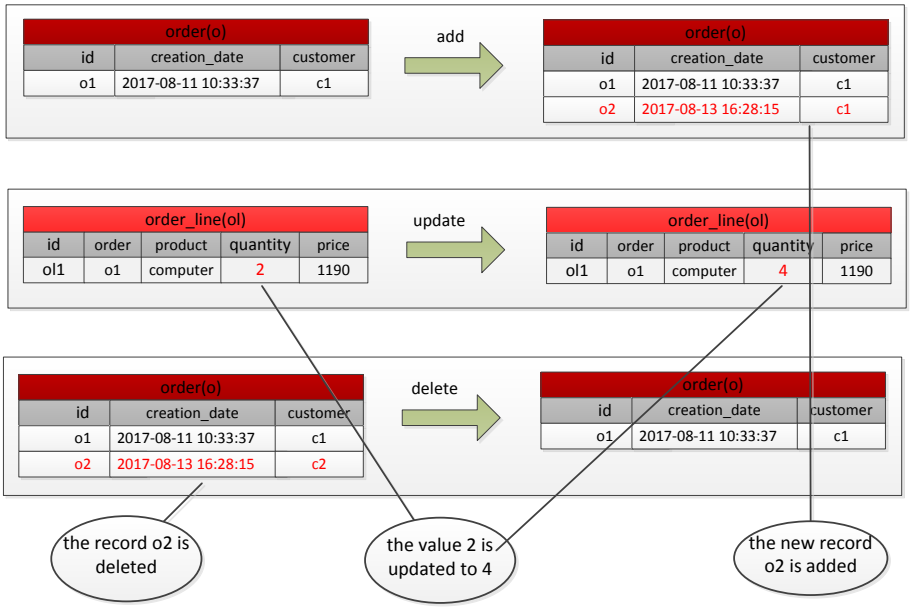


Figure 3.14: Different types of changes.

attributes.

Definition 3.11 (Change Types) *Let* $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ *be a data model.* $CT^{DM} = \{(op, c, Attr')\} = CT^{DM}_{add} \cup CT^{DM}_{upd} \cup CT^{DM}_{del}$ *is the set of change types composed of the following pairwise disjoint sets:*

- $CT^{DM}_{add} = \{(\oplus, c, Attr') \mid c \in C \wedge Attr' \subseteq classAttr(c) \wedge keyAttr(classPK(c)) \subseteq Attr'\}$ *are the change types for adding objects of DM,*
- $CT^{DM}_{upd} = \{(\oslash, c, Attr') \mid c \in C \wedge Attr' \subseteq classAttr(c) \wedge keyAttr(classPK(c)) \cap Attr' = \emptyset\}$ *are the change types for updating objects of DM, and*
- $CT^{DM}_{del} = \{(\ominus, c, Attr') \mid c \in C \wedge Attr' = \emptyset\}$ *are the change types for deleting objects of DM.*

$\mathcal{U}_{CT}$ *is the universe of change types.*

Based on the discussion about differences of changes, a change type is formalized as a tuple of an operation (sentence) type *op*, a class *c* and a set of

attributes $Attr'$. Note that $Attr'$ can only contain attributes of the class c (i.e., $Attr' \subseteq classAttr(c)$). Besides, it should satisfy different requirements for different operation types. If $op = \oplus$, $Attr'$ must include all attributes of the primary key (i.e., $keyAttr(classPK(c)) \subseteq Attr'$), as the primary key columns must be instantiated when a record is inserted into databases. In contrast, $Attr'$ cannot include any attribute of the primary key (i.e., $keyAttr(classPK(c)) \cap Attr' = \emptyset$) when $op = \oslash$, since the primary key values of a record cannot be updated. If $op = \ominus$, $Attr'$ is an empty set as no attributes are instantiated or updated. Note that $Attr'$ only indicates the instantiated or updated attributes, and it is not mandatory to contain primary key attributes for access (which is provided later).

For instance, the change types of the three changes in Figure 3.14 can be specified as $(\oplus, order, \{id, creation\_date, customer\})$, $(\oslash, order\_line, \{quantity\})$ and $(\ominus, order, \{\})$, respectively.

Each database change can be considered as an instance of a change type, which impacts the state of a record, indicated by all attributes of the record. More precisely, a change inserting a record means that all attributes of the record are instantiated and assigned with values. A change updating a record means that some attributes of the record are updated. A change deleting a record means that all attributes of the record are inexistent. Next, we use two mapping functions to describe the change of attributes to formalize changes.

Definition 3.12 (Changes) *Let DM be a data model, $CT^{DM} = CT^{DM}_{add} \cup CT^{DM}_{upd} \cup CT^{DM}_{del}$ be the set of change types and $\perp$ be the null value. $CH^{DM} = CH^{DM}_{add} \cup CH^{DM}_{upd} \cup CH^{DM}_{del}$ is the set of changes composed of the following pairwise disjoint sets:*

- $CH^{DM}_{add} = \{(\oplus, c, Attr', map_{old}, map_{new}) \mid (\oplus, c, Attr') \in CT^{DM}_{add} \wedge map_{old} = \perp \wedge (c, map_{new}) \in O^{DM} \wedge (\forall attr \in Attr' : map_{new}(attr) \neq NULL) \wedge (\forall attr \in classAttr(c) \setminus Attr' : map_{new}(attr) = NULL)\}$,
- $CH^{DM}_{upd} = \{(\oslash, c, Attr', map_{old}, map_{new}) \mid (\oslash, c, Attr') \in CT^{DM}_{upd} \wedge (c, map_{old}) \in O^{DM} \wedge (c, map_{new}) \in O^{DM} \wedge (\forall attr \in classAttr(c) \setminus Attr' : map_{new}(attr) = map_{old}(attr))\}$, *and*
- $CH^{DM}_{del} = \{(\ominus, c, Attr', map_{old}, map_{new}) \mid (\ominus, c, Attr') \in CT^{DM}_{del} \wedge (c, map_{old}) \in O^{DM} \wedge map_{new} = \perp\}$.

A *change* $(op, c, Attr', map_{old}, map_{new})$ corresponds to an SQL sentence, i.e., adding, updating or deleting a record in a table. Its *change type* $(op, c, Attr')$ indicates which table (i.e., c) the record is in, which columns (i.e., $Attr'$) of the record are impacted and how the record is impacted (indicated by op, i.e., adding, updating or deleting). The old and new mappings (i.e., map_{old} and map_{new})

specify the contents of the record before and after the change, respectively. Note that map_{old} and map_{new} should contain all primary key values for accessing the impacted records. $(c, \perp)$ indicates an inexistent record in the database.

For instance, the top change in Figure 3.14 can be specified as $(\oplus, order, \{id, creation\_date, customer\}, \perp, map)$ where $map(creation\_data) = 2017-08-11\ 10:33:37$, $map(id) = o1$ and $map(customer) = c1$. Table 3.4 shows the formalization of the three changes in Figure 3.14 (the "creation_date" attribute only presents the date for simplicity). Note that $Attr'$ provides more power to produce different change types, i.e., changes impacting different attributes in the same table can be classified into different change types, which is not considered in [51].

Index	Type	Class	Attributes	OldMap	NewMap	Timestamp
1	⊕	order	id, creation_date, customer	map_{null}	id:ol, creation_date:2017-08-11, customer:c1	ts_1
2	⊘	order_line	quantity	id:oll, order:o1, product:computer, quantity:2, price:1190	id:oll, order:o1, product:computer, quantity:4, price:1190	ts_2
3	⊖	order	id, creation_date, customer	id:ol, creation_date:2017-08-11, customer:c2	map_{null}	ts_3

Table 3.4: A fragment of change log.

Definition 3.13 (Change Occurrence, Change Log) *Let DM be a data model and CH^{DM} be the set of changes of DM. Let $\mathcal{U}_{TS}$ be the universe of timestamps. $CO^{DM} = CH^{DM} \times \mathcal{U}_{TS}$ is the set of all possible change occurrences of DM. A change log $CL = \langle co_1, co_2, ..., co_n \rangle \in (CO^{DM})^*$ is a sequence of change occurrences such that time is non-decreasing, i.e., $ts_i \leqslant ts_j$ for any $co_i = (ch_i, ts_i)$ and $co_j = (ch_j, ts_j)$ with $1 \leqslant i < j \leqslant n$.*

A *change occurrence* $co = (ch, ts)$ represents a change ch happening at ts. It corresponds to one row in the redo log or CDHDR table. A *change log* consists of a list of change occurrences which are sorted by timestamps such that time is non-decreasing. Table 3.4 presents a fragment of a change log containing three change occurrences. Note that a change log records events on the database level, i.e., it can be considered as an event log on the database level. Next, we explain the events at the information system level.

3.2.3 Events at the Information System Level

In Section 3.2.2, we defined change occurrences, i.e., events at the database level. Note that the change occurrences are atomic and they are different from events on information systems, since one event may result in multiple change occurrences in different tables. For instance, one click on the "create order" button may insert one record in the "order" table and two records in the "order_line" table as shown in Figure 3.15. In other words, one "create order" event results in three change occurrences. In this part, we lift events from the database level (i.e., change occurrences) to the information system level.

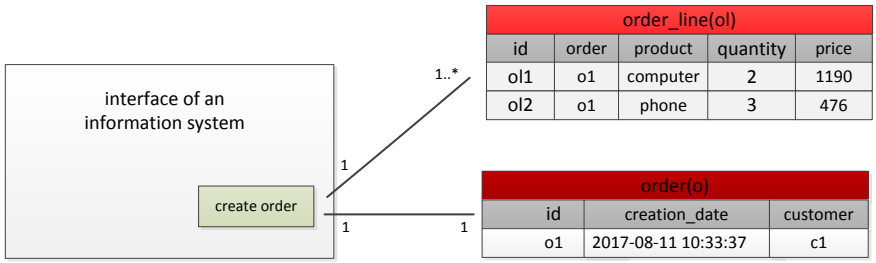


Figure 3.15: One event in an information system corresponds to multiple changes in different tables.

Here, we assume that the change occurrences corresponding to one event have the same timestamp. For instance, in Figure 3.15 the three change occurrences $co_1 = (ch_1, ts_1)$, $co_2 = (ch_2, ts_2)$ and $co_3 = (ch_3, ts_3)$ that correspond to the "create order" event should have the same timestamp, i.e., $ts_1 = ts_2 = ts_3$. Based on this idea, events can be defined as follows, i.e., events are like change logs at a particular time.

Definition 3.14 (Events) *Let DM be a data model, CO^{DM} be the set of change occurrences of DM. An event $e = \langle co_1, co_2, ..., co_n \rangle \in (CO^{DM})^*$ is a sequence of change occurrences which have the same timestamp, i.e., $\forall (ch_i, ts_i), (ch_j, ts_j) \in e : ts_i = ts_j$. E^{DM} is the set of events of DM*

As shown in Definition 3.14, an event is a sequence of change occurrences which have the same timestamp. For instance, $e = \langle ((\oplus, o, Attr, map_{old}, map_{new}), ts), ((\oplus, ol, Attr', map'_{old}, map'_{new}), ts), ((\oplus, ol, Attr', map''_{old}, map''_{new}), ts) \rangle$ is an event consisting of three changes happening at ts. It corresponds to the "create order" event in Figure 3.15. Based on the definition of events, a change log can be transformed into a sequence of events, by grouping changes into events.

Definition 3.15 (Transforming Change Log into Event Sequence) *extractES* $\in (CO^{DM})^* \rightarrow (E^{DM})^*$ *is a function to extract an event sequence from a change log such that* $\forall CL = \langle co_1, co_2, ..., co_n \rangle \in (CO^{DM})^* : extractES(CL) = \langle e_1, e_2, ..., e_m \rangle$ *where*

- $\langle co_1, co_2, \ldots, co_n \rangle = e_1 \oplus e_2 \oplus \ldots \oplus e_m$,
- $\forall 1 \leq i \leq m : \forall (ch, ts), (ch', ts') \in e_i : ts = ts'$, *and*
- $\forall 1 \leq i < j \leq m : \forall (ch, ts) \in e_i, (ch', ts') \in e_j : ts < ts'$.

Change Log		Event Sequence	
Index	Change Occurrence	Index	Event
1	$co_1 = (ch_1, ts_1)$	1	$e_1 = \langle co_1 \rangle$
2	$co_2 = (ch_2, ts_2)$	2	$e_2 = \langle co_2, co_3, co_4 \rangle$
3	$co_3 = (ch_3, ts_2)$		
4	$co_4 = (ch_4, ts_2)$		
5	$co_5 = (ch_5, ts_3)$	3	$e_3 = \langle co_5, co_6 \rangle$
6	$co_6 = (ch_6, ts_3)$		

Table 3.5: Transforming a change log into an event sequence.

Function *extractES* transforms a change log (i.e., a change occurrence sequence) into an event sequence by grouping change occurrences at the *same time* into an event without modifying the order of the change occurrences. For instance, Table 3.5 shows an example of transforming a change log with six change occurrences into an event sequence with three events. Note that it is possible to group change occurrences based on *domain knowledge* instead of *time*, which provides more freedom for extracting events (e.g., overlapping events).

Note that the assumption that change occurrences of the same timestamp correspond to one event may be problematic in some situations. The assumption is impacted by the granularity of the timestamp. In practice, it is possible that multiple events make changes with the same timestamp. Here, we just give a naive approach based on the assumption. More advanced approaches can be proposed as future work. Another solution to avoid the problem is to consider a change occurrence as an event, without distinguishing events at database or information system levels [51, 135].

3.3 Connecting the Behavioral Perspective to the Data Perspective

Section 3.1 introduced the data perspective of a data set, which abstracts the data structure and data elements. Section 3.2 explained the behavioral perspective, i.e., extract and formalize events on the database level and information system level. Note that there exist interactions between these two perspectives, e.g., events from the behavioral perspective may modify objects from the data perspective. In this section, we define a notion of *object-centric event data* to combine these two perspectives [88].

Before integrating these two perspectives, we investigate their interactions. Originally, the interactions are derived from the fact that database changes impact objects. Consider for example a change occurrence $((op, c, Attr', map_{old}, map_{new}), ts)$. It changes an object at the time ts and the two mapping functions map_{old} and map_{new} indicate the values of attribute of the object before and after the change, respectively. Based on the functions, we can find the object which is impacted by the database change. In the following, we define a function to connect database changes to the corresponding objects.

Definition 3.16 (Relating Changes to Objects) *Let DM be a data model and $co = ((op, c, Attr', map_{old}, map_{new}), ts) \in CO^{DM}$ be a change occurrence. Function $change \in CO^{DM} \rightarrow O^{DM}$ gives the object changed by each change occurrence, such that $change(co) = (c, map)$ where*

- *if $op \neq \ominus$, $map = map_{new}$, or*
- *if $op = \ominus$, $map = map_{old}$.*

Each database change from the behavioral perspective is related to an object from the data perspective. When the change adds or updates an object, the new (i.e., the added or updated) object is its related object. In contrast, when the change deletes an object, the old (i.e., deleted) object is its related object. For instance, the first change in Table 3.4 is related to the object $o1$ which represents the first record in the "order" table in Figure 3.3.

Definition 3.17 (Relating Events to Objects) *Let DM be a data model and $E \subseteq E^{DM}$ be a set of events. Function $relate \in E \rightarrow \mathcal{P}(O^{DM})$ gives the objects impacted by an event $e \in E$, such that $relate(e) = \{change(co) \mid co \in e\}$.*

As shown in Definition 3.14, one event may contain multiple change occurrences. Accordingly, one event may be related to multiple objects, since each

change occurrence is related to one object and there may be multiple. Based on the function *change*, function *relate* returns the objects related to each event. For instance, the "create order" event described in Figure 3.15 is related to three objects, $o1$, $ol1$ and $ol2$. After correlating events from the behavioral perspective to objects from the data perspective, we define object-centric event data to combine these two perspectives.

Definition 3.18 (Object-Centric Event Data) *Let DM be a data model. Object-centric event data are represented by a tuple* $OCED = (OM, ES, relate)$*, where*

- $OM = (Obj, Rel, class, attrO)$ *is a valid object model of DM extracted from database elements (i.e., table records), representing the data perspective,*
- $ES = extractES(CL)$ *is an event sequence extracted from a change log CL, representing the behavioral perspective, and*
- $relate \in E \rightarrow \mathscr{P}(O^{DM})$ *describes the interactions between two perspectives, where* $E = \{e \in ES\}$.

$\mathscr{U}_{OCED}$ *is the universe of object-centric event data sets.*

An object-centric event data set $OCED = (OM, ES, relate)$ integrates information derived from database tables and other files, such as redo logs and change tables. The object model OM describes the records and dependencies between records in tables. It can be considered as the snapshot of the database after all operations recorded in a change log CL. The event sequence extracted from CL describes all operations on the database, which result in the object model OM. Function *relate* builds a loose connection between the data and behavioral perspectives by correlating events to objects. Note that the range of *relate* is a set $Obj' \in \mathscr{P}(O^{DM})$ rather than Obj. It means that an event e can refer to an object $o \notin Obj$, since e may modify o at some point in time and then o is deleted from the database (i.e., OM).

Table 3.6 shows an object-centric event data set, extracted from the database tables in Figure 3.3 and related redo logs in Table 3.1. The "Object Model" column describes the object model OM. The "Event" column shows all events in the event sequence ES, and the order of the events is shown in the "Index" column, which are derived based on the timestamps in the "Timestamp" column. Figure 3.16 provides a visualization solution for the object-centric event data in Table 3.6. The object model OM is located at the bottom, whose objects are depicted as grey dots and clustered in classes, e.g., $o1$ is of "order" class. The event sequence E is shown on the top. For simplicity, we hide some events and objects. The dotted lines in the middle show the interactions between events and objects. For instance, $e1$ refers to objects $ol1$, $ol2$ and $o1$.

Index	Event	Timestamp	Related Objects	Object Model	
				Objects	Relations
1	e1	2017-08-11 10:33:37	o1,ol1,ol2	c1,c2,o1,	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),
2	e2	2017-08-13 16:28:15	o2,ol3,ol4	ol1,ol2, o2,	(r10,o2,ol3),(r10,o2,ol4),(r4,c1,s1),(r6,s1,sl1),
3	e3	2017-08-14 11:36:35	s1,sl1	ol3,ol4,s1,	(r9,ol1,sl1),(r1,c1,i1),(r3,i1,er1),(r8,o1,er1),
4	e4	2017-08-15 09:13:27	i1,er1	sl1,i1,er1,	(r4,c1,s2),(r6,s2,sl2),(r9,ol1,sl2),(r6,s2,sl3),
5	e5	2017-08-16 14:15:31	s2,sl2,sl3	s2,sl2,sl3,	(r9,ol2,sl3),(r1,c1,i2),(r3,i2,er2),(r8,o1,er2),
6	e6	2017-08-17 17:38:36	i2,er2,er3	i2,er2,er3,	(r3,i2,er3),(r8,o2,er3),(r4,c1,s3),(r6,s3,sl4),
7	e7	2017-08-19 13:22:04	s3,sl4,sl5	s3,sl4,sl5,	(r9,ol3,sl4),(r6,s3,sl5),(r9,ol4,sl5),(r7,p1,pl1),
8	e8	2017-08-21 16:26:13	p1,pl1,pl2	p1,pl1,pl2,	(r2,i1,pl1),(r7,p1,pl2),(r2,i2,pl2),(r1,c1,i3),
9	e9	2017-08-23 14:23:19	i3,er4	i3,er4,p2,	(r3,i3,er4),(r8,o2,er4),(r7,p2,pl3),(r2,i3,pl3)
10	e10	2017-08-26 14:53:49	p2,pl3	pl3	

Table 3.6: Details of an object-centric event data set.

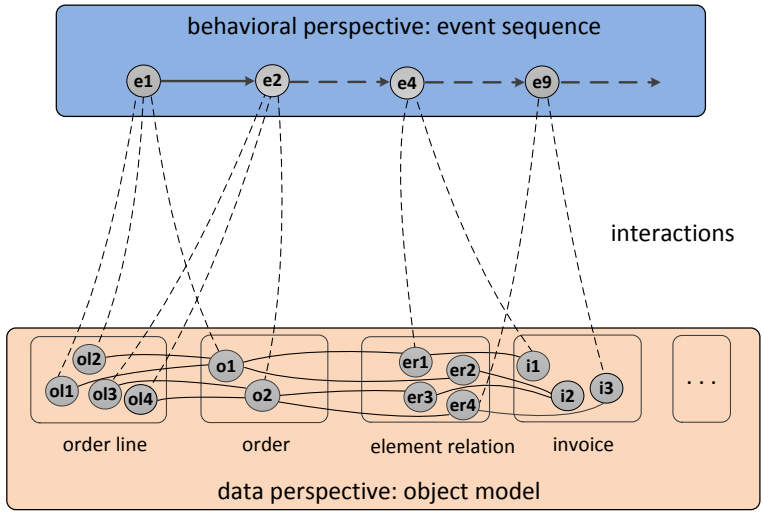


Figure 3.16: Object-centric event data combining data and behavioral perspectives, by relating events in the event sequence to objects in the object model.

In object-centric event data, each event has some related objects, indicated by the function *relate*. Note that it is possible that an object o related to an event $e \in E$ is not in the object model, i.e., $o \notin OM$ (e.g., removed objects). Consider the next situation to understand why some objects may not exist in the object model. For instance, a "create order" event e adds an "order" object o in a database at time t_0, i.e., $o \in relate(e)$. Then a "delete order" event removes o at

time t_1 ($t_1 > t_0$). After these events, we collect data and derive an object model OM representing the state of the database at time t_2 ($t_2 > t_1$). Apparently, the removed "order" object o is not contained by OM, i.e., $o \notin OM$.

In summary, an object-centric event data set describes the events operated on databases and the resulting object model. The object model can be considered as the current state of the database, i.e., at some point in time when we derive the database tables. In the context of an XOC log, this model corresponds to the last event in the log. In next chapter, we will reconstruct object models for each event based on object-centric event data, resulting in an XOC log.

3.4 Features of Object-Centric Event Data

Up to now, all elements involved in object-centric event data have been illustrated. In general, as indicated by the name "object-centric event data", such data consist of two perspectives, i.e., data perspective (indicated by "object") and behavioral perspective (indicated by "event"). Besides, there exist interactions between these two perspectives. The structure of object-centric event data conforms to the structure of artifact-centric information systems where (i) database techniques are employed to store transactions as table records, (ii) events (operations on information systems) are recorded implicitly in redo logs and change tables, and (iii) events on information systems modify, i.e., add, update or delete table records. In this section, we provide more details about the features of object-centric event data from different angles.

Object-centric. Object-centric event data are extracted from various sources related to artifact-centric information systems. Their major feature is "object-centric", since (i) they are generated by information systems which are artifact-centric (or module-centric), (ii) they are stored in database tables which are table-centric, and (iii) the data perspective serves as the backbone for the whole data (i.e., data-centric), which are different from "case-centric" data (generated by WFM/BPM systems) where the behavioral perspective is the main focus. In summary, the semantics of "object-centric" is the combination of "artifact-centric" source, "table-centric" storage and "data-centric" focus.

No case notions. Object-centric event data are generated by executing business processes on artifact-centric information systems. Note that each system consists of multiple artifacts and each artifact has its own case notion (artifact id) for the life-cycle (i.e., sub-process) related to it. The object-centric event data collected from the process often contain data from multiple artifacts, such that the whole data set has multiple case notions. As a result, it is difficult to identify a global

case notion for the whole data.

Events are merely changes of the object model. In BPM/WFM systems, events are explicitly recorded in event logs. In contrast, artifact-centric information systems employ databases to store transactions, which do not record events explicitly. Fortunately, transactions leave footprints in various files such as redo logs, which can be exploited to extract events. However, in this way, the extracted events are scattered across various sources, and in many cases, there is no well-documented information on how events are related to each other and to the overall business process. Therefore, the derived events can only form a weak behavioral perspective and the analysis of the behavioral perspective needs support from the data perspective (e.g., the scattered events can be correlated through objects from the data perspective).

Presence of one-to-many and many-to-many relationships. Often there exist one-to-many and many-to-many relationships between different artifacts in artifact-centric information systems. For instance, the OTC business process supported by ERP systems involves multiple artifacts, such as order, invoice and shipment. In the process, one order may be split into multiple shipments and one shipment may contain products from multiple orders. As a result, there exists a many-to-many relationship between the order artifact and shipment artifact. Accordingly, the object-centric event data generated in the process also have such complex relationships.

3.5 Summary

In this chapter, we defined the notion of object-centric event data, which is a novel structure to organize the data generated by artifact-centric information systems. It integrates the data perspective, behavioral perspective and the interactions between these two perspectives. From the data perspective, a data model is employed to describe database schema and an object model is defined to represent database records. From the behavioral perspective, events at the database level and information system level are defined to abstract the operations on information systems. Besides, we discussed the features of object-centric event data, and compared them with traditional case-centric data.

Conventional process mining algorithms have difficulties dealing with object-centric event data. Consider for example many-to-many relationships. Traditional process notations, such as BPMN, EPCs, and UML activity diagrams, model the life cycle of one type of instances in isolation. They cannot properly describe the many-to-many relationships between different types of instances. Besides,

there is a lack of process mining techniques able to discover process models with many-to-many relationships. Therefore, after introducing object-centric event data, we propose new process mining techniques to better deal with such data in next chapters.

Chapter 4

eXtensible Object-Centric Event Logs

As discussed, process mining helps organizations to investigate how their business processes are executed and how they can be improved based on event logs extracted from information systems. The eXtensible Event Stream (XES) format is the current event log standard, which requires a case notion to correlate events (cf. Chapter 2). However, it has problems to deal with object-centric event data (e.g., database tables) due to the existence of one-to-many and many-to-many relations (cf. Chapter 3). In order to enable process mining on object-centric event data, in this chapter, we define a novel log format, resulting in eXtensible Object-Centric (XOC) event logs. Besides, an approach is proposed to extract XOC logs from object-centric event data, which shows that such event logs can indeed be extracted from today's information systems.

Figure 4.1 positions Chapter 4 in the whole framework of the thesis. This chapter is the second step of the extraction part (highlighted in red). It is based on the result derived in Chapter 3. Figure 4.2 presents the difference between object-centric event data and XOC logs. More precisely, object-centric event data have only one object model (representing the current state of database tables) while each event in XOC logs has a corresponding object model (representing the state of database tables just after the event). Besides, Figure 4.2 also revels the idea to transform object-centric event data into XOC logs. Based on the current object model, previous object models are reconstructed for all events in XOC logs.

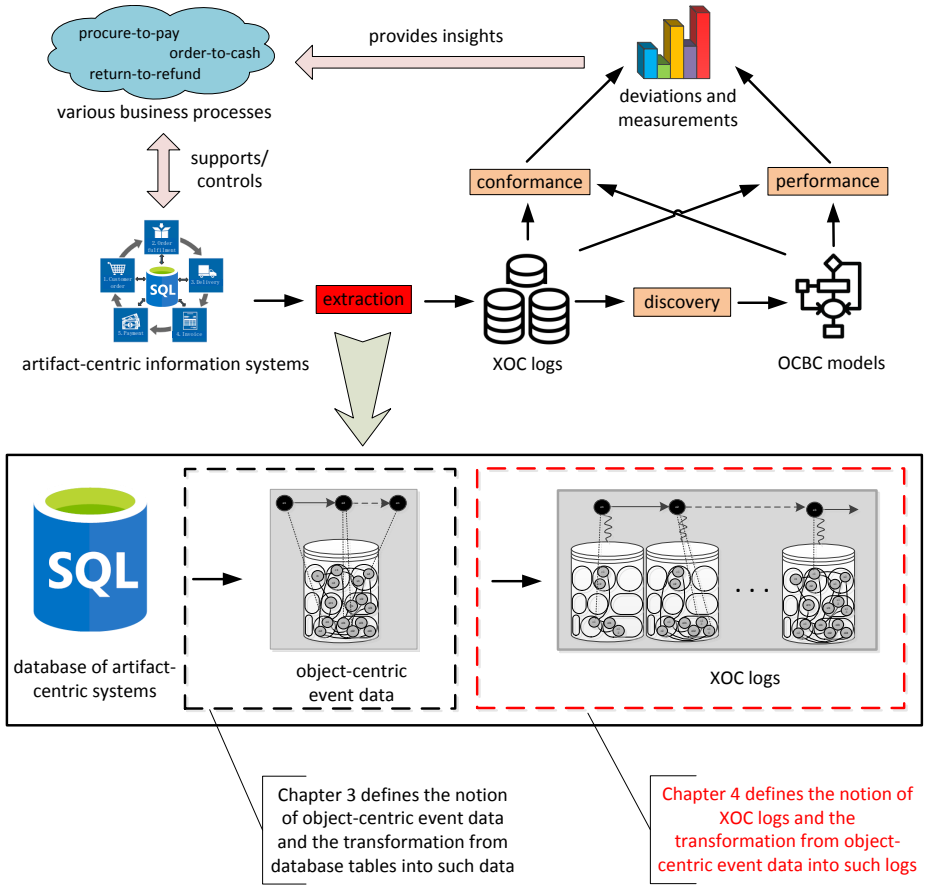


Figure 4.1: Positioning Chapter 4 in the whole framework.

This chapter is organized as follows. In Section 4.1, we motivate our approach by discussing the challenges encountered by conventional log formats when they are applied to object-centric event data. The XOC event logs are formally defined in Section 4.2 and Section 4.3 presents an approach to extract XOC logs from object-centric event data. We evaluate XOC logs by comparing them to XES logs in Section 4.4. Section 4.5 reviews the related work while Section 4.6 concludes this chapter.

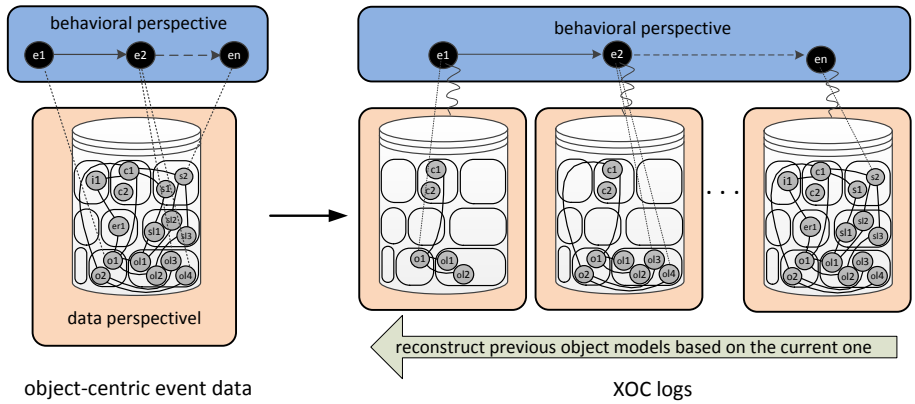


Figure 4.2: Transforming object-centric event data into XOC logs.

4.1 Challenges Encountered by Conventional Log Formats

Process mining represents a set of techniques which are widely used to extract insights from event data generated by information systems. The starting point for process mining techniques is formed by event logs. The XES log format [1] is widely employed. In general, an XES log consists of a collection of traces. A trace describes the life-cycle of a particular case (i.e., a process instance) in terms of the activities executed (cf. Chapter 2). Note that a case notion is mandatory for XES logs, which is used to correlate events into traces.

However, the information systems one encounters in most organizations are *artifact-centric*, such as ERP and CRM systems, which do not assume a case notion. A common feature of these systems is that they are built on top of database technology, i.e., they contain hundreds of tables covering customers, orders, deliveries, etc. There exist one-to-many and many-to-many relationships between different tables. In addition, the events generated by these systems are often recorded implicitly, e.g., events are stored as changes in redo logs or timestamp values in tables.

As the nature of data generated by artifact-centric information systems does not match the constraints set by the XES format (e.g., a mandatory case notion), the following challenges have been identified when applying the XES format to object-centric event data:

- *It is difficult to identify the case id for the whole process*. Object-centric event data lack a single explicit definition of case notion for the end-to-end process. On the contrary, because of the existence of multiple classes of objects (tables) and relations (foreign keys), events in object-centric event data can be correlated in many ways. These correlations may correspond to different processes affecting the same data, or different views on the same process (customer vs. provider point of view). In the case of XES, a case notion has to be defined beforehand to proceed with the extraction, requiring one dedicated event log for each perspective to be analyzed.
- *The quality of the input data gets compromised*. If we straightjacket object-centric event data into XES logs, the input data with one-to-many and many-to-many relations are flattened into separate traces, in which events referred to by multiple cases are duplicated (cf. Figure 4.13). This forced transformation introduces unnecessary redundancy and leads to problems such as data convergence and divergence [93] (cf. Section 4.4.4).
- *Interactions between process instances get lost*. Traces in XES logs only describe the evolution (i.e., lifecycle) of one type of process instance and they are typically considered in isolation. Due to the lack of interactions between process instances, XES logs cannot provide a whole view to indicate the state of a system.
- *The data perspective is only secondary*.[1] The XES format focuses on the behavioral perspective, considering any additional information as trace or event attributes. In this sense, the information not directly related to the events is discarded (records, data objects, etc.), which weakens the data perspective of the original system.

To address the problems mentioned above, we propose a novel log format to organize object-centric event data from databases, resulting in *eXtensible Object-Centric (XOC)* logs. This log format provides an evolutionary view on databases based on the idea that a log consists of a list of events and each event refers to an *object model* representing the state of the database just updated by the event.

[1] The behavioral perspective refers to the control-flow of events while the data perspective refers to data attributes.

4.2 eXtensible Object-Centric (XOC) Event Log Format

In order to solve the problems encountered by XES logs, we define a novel event log format, i.e., XOC, to organize the object-centric event data in this section. More precisely, we (i) explain the basic idea of the XOC format, (ii) define the XOC format by formalization, and (iii) present a concrete XML serialization of XOC logs.

4.2.1 Principles of Artifact-Centric Information Systems

Before diving into the concrete log format, we analyze the principles followed by artifact-centric information systems, summed up as follows:

- *events* (i.e., operations) on information systems change the states of business processes on these systems;
- *the states of business processes* can be reflected (or represented) by database contents, i.e., records in database tables.

Furthermore, we use a concrete example to explain the principles. Figure 4.3 presents the evolution of states of a process along with events. The process is a variant of the order-to-cash process, which consists of five states (or activities). The highlighted state in red indicates the state of the process just after the

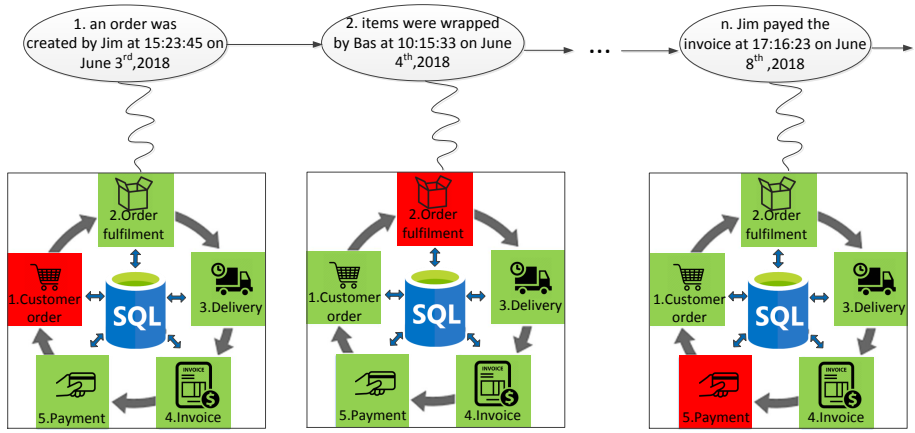


Figure 4.3: Events change the states of (processes on) information systems.

occurrence of the corresponding event. For instance, after the first event ("an order was created by Jim at 15:23:45 on June 3rd, 2018"), the process is in the "Customer order" state, which means that an order is created in the process. Similarly, after n events, the process locates in the "Payment" state.

When the events are executed in the process, all the generated transactions are stored in a relational database (indicated by the "SQL" cylinder in the center of the business process in Figure 4.3). The records in the database tables reflect the highlighted state in red. For example, after the first event that Jim created an order at 15:23:45 on June 3rd, 2018, an "order" record is added in the "order" table, which indicates that the process is in the state that an order is created (i.e., "Customer order"). Note that the state may depend on the contents of multiple records. For instance, if there exists an "order" record in the "order" table and a corresponding "payment" record in the "payment" table, it indicates that the process is located in the "Payment" state.

Based on the principles explained above, the basic idea of the novel log format, i.e., XOC, is to record operations on artifact-centric systems and the corresponding state of the process in the system after each operation. More precisely, an event log consists of a list of events which represents operations, and each event corresponds to an object model which represents the state of the process just after the execution of the event. According to this idea, we define the XOC log format in the next section.

4.2.2 Log Definition

In general, event logs are used to record past events related to information systems. Based on the basic idea explained in Section 4.2.1, an XOC event log is a collection of events that belong together without assuming some case for the whole process, i.e., they belong to some "process" where many types of objects may interact.

Each event corresponds to an *activity* and may have additional *attributes* such as the time at which the event took place, the resource executing the corresponding event, the type of event (e.g., start, complete, schedule, abort), the location of the event, or the costs of an event. Each event attribute has a *name* (e.g., "age") and a *value* (e.g., "49"). To model the overlapping of activities in time one can use start and complete events.

Besides, each event corresponds to an object model and the event refers to objects in the object model. Note that one event may refer to multiple objects and one object may be referred to by multiple events. The objects referred to by

an event indicate they are modified by the operation corresponding to the event. Moreover, events are atomic and ordered. For simplicity, we assume a *total order*.

Definition 4.1 (eXtensible Object-Centric (XOC) Event Log) *Let DM be a data model. An* event log *is a tuple* $L = (E, act, attrE, relate, om, \preceq)$*, where*

- $E \subseteq \mathcal{U}_E$ *is a set of events,*
- $act \in E \to \mathcal{U}_A$ *maps events onto activities,*
- $attrE \in E \to (\mathcal{U}_{Attr} \not\to \mathcal{U}_{Val})$ *maps events onto a partial function assigning values to some attributes,*
- $relate \in E \to \mathcal{P}(O^{DM})$ *relates events to sets of object references,*
- $om \in E \to \mathcal{U}_{OM}$ *maps each event to the object model directly after the event took place, and*
- $\preceq \subseteq E \times E$ *defines a total order on events.*

$\mathcal{U}_L$ *is the universe of XOC event logs.*

An XOC log $L = (E, act, attrE, relate, om, \preceq)$ consists of a set of events E with a total order defined by $\preceq$. Each event has a corresponding activity and some attributes, specified by the functions *act* and *attrE*, respectively. In order to relate the behavioral perspective and the data perspective (i.e., events and objects), each event refers to at least one object, indicated by *relate*. Moreover, each event is associated with an object model indicated by *om*, which represents the state of the process just after the execution of the event.

Table 4.1 presents an XOC log example. More precisely, the numbers in the "Index" column indicate the order of events (e.g., $co1 \preceq co2$ and $co2 \preceq cs1$). In order to refer to a specific event, the "Event" column assigns an ID to each event, which specifies the set of events, i.e., $E = \{co1, co2, cs1, ci1, cs2\}$. The function *act* is indicated by the "Activity" column, e.g., $act(co1) =$ *create order* (cf. Section 4.3 for details to derive *act*). Besides, each event may have additional attributes. For example, the fourth event $ci1$ has two attributes, "create time = 2017-08-15" (we only show the date for simplicity), and "amount = 2380". The function *relate* is indicated by the "Reference" column, e.g., $ci1$ refers to $i1$ and $er1$ (i.e., $relate(ci1) = \{i1, er1\}$). The object model of each event is in the "Object Model" column. For instance, $om(co1) = OM_{co1} = (Obj_{co1}, Rel_{co1}, class_{co1}, attrO_{co1})$ where $Obj_{co1} = \{c1, c2, o1, ol1, ol2\}$ and $Rel_{co1} = \{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2)\}$. Note that each object has a unique ID, a corresponding class and additional attributes. In the table, the class is indicated by its ID while the additional attributes are omitted. For instance, $ol1$ indicates its corresponding class is ol (denoting "order line"). In this thesis, we refer directly to Obj_e, Rel_e, $class_e$, $attrO_e$ for $e \in E$ if the context is clear.

Index	Event	Activity	Attributes	References	Object Model	
					Objects	Relations
1	co1	create order	create date=2017-08-11	o1,ol1,ol2	c1,c2,o1,ol1,ol2	(r5,c1,o1), (r10,o1,ol1), (r10,o1,ol2)
2	co2	create order	create date=2017-08-13	o2,ol3,ol4	c1,c2,o1,ol1,ol2, o2,ol3,ol4	(r5,c1,o1),(r10,o1,ol1), (r10,o1,ol2), (r5,c1,o2), (r10,o2,ol3),(r10,o2,ol4)
3	cs1	create shipment	create date= 2017-08-14	s1,ol1	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1	(r5,c1,o1),(r10,o1,ol1), (r10,o1,ol2), (r5,c1,o2), (r10,o2,ol3),(r10,o2,ol4), (r4,c1,s1),(r6,s1,sl1), (r9,ol1,sl1)
4	ci1	create invoice	create date=2017-08-15, amount=2380	i1,er1	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1	(r5,c1,o1), (r10,o1,ol1), (r10,o1,ol2), (r5,c1,o2), (r10,o2,ol3),(r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1),(r8,o1,er1)
5	cs2	create shipment	create date=2017-08-16	s2,ol1,ol2	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2), (r5,c1,o2), (r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1),(r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1), (r8,o1,er1), (r4,c1,s2), (r6,s2,sl2), (r9,ol1,sl2), (r6,s2,sl3), (r9,ol2,sl3)

Table 4.1: An example of an XOC log.

Figure 4.4 shows a graph to present the XOC log in Table 4.1. More precisely, the black dots at the top denote the events (the IDs of events are shown in the center) while the cylinders at the bottom represent object models. There are rounded rectangles in the cylinders, which cluster the objects of the same classes. The curves between black dots and cylinders indicate the correspondence between events and object models. The dotted lines specify the object references for each event and the solid lines correspond to the object relations between objects. The arrows between black dots indicate the order between events. For instance, the event $co1$ is denoted by the first black dot and its corresponding object model is represented by the first cylinder. The object model has 5 objects of 3 classes, and 3 object relations. $co1$ refers to three objects highlighted in red, i.e., $o1$, $ol1$ and $ol2$.

Figure 4.4 also illustrates the evolution of the object model. After the occurrence of an event, objects and object relations may have been added, updated (i.e., the attribute values) or deleted. This log format provides an evolution view of the process in an information system based on the idea that events of a process executed on an information system change the state (i.e., adding, updating or deleting records) of the underlying database.

The event log provides a *snapshot of the object model after each event*. This triggers the question: Can the object model be changed in-between two subsequent events? If no such changes are possible, then the object model before an event is the same as the object model after the previous event. If we would like to allow for updates in-between events, then these could be recorded in the log. Events referring to some artificial activity *update* could be added to signal

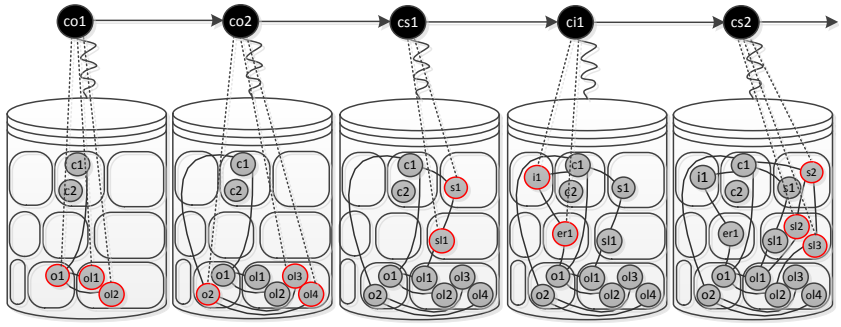


Figure 4.4: A “graphical” representation for the XOC log in Table 4.1, revealing the evolution of a database.

the updated object model. We could also explicitly add a snapshot of the object model just before each event. In the remainder, we only consider the snapshot OM_e after each event $e \in E$. In other words, we assume that object models cannot be changed in-between two subsequent events.

Note that Definition 4.1 calls for event logs different from the standard *XES format*. XES (`www.xes-standard.org`), which is supported by the majority of process mining tools, assumes a case notion (i.e., each event refers to a process instance) and does not keep track of object models. In the next section, we discuss the meta model of XOC logs and give a concrete XML serialization for XOC log files.

4.2.3 XOC Log Files

In this section, we employ an XOC meta model to describe how the elements in XOC logs (such as events, object models and attributes) are organized in concrete XOC log files. The idea here is to refer to the existing XES meta model and file structure [136], removing elements related to "case" and adding necessary elements related to "object".

Figure 4.5 shows the XOC meta model expressed in terms of a UML class diagram. The dark rounded squares describe the backbone of an XOC log while the light rounded squares describe additional contents (e.g., attributes). It consists of the following classes:

- *Log*. An XOC log consists of any number of events (indicated by the cardinalities on the edge between "Log" and "Event").
- *Event*. Each event describes an operation on the information system and corresponds to an object model which represents the state of the system just after the event. Each event also refers to a set of objects (i.e., "References") to indicate which objects are modified by the event, and to a set of attributes to indicate the properties of the event (e.g., activity and resource).
- *Object model*. One object model consists of a set of objects (i.e., "Objects") and a set of object relations (i.e., "Relations"). Note that the object or relation sets can be empty. In the object model, each object relation connects two objects.
- *Attribute*. The log, events, objects and object relations may have attributes. Attributes may be nested. There are five core types: "String", "Date", "Int", "Float", and "Boolean". These correspond to the standard XML types: "xs:string", "xs:dateTime", "xs:long", "xs:double", and "xs:boolean", respectively. For example, "2011-12-17T21:00:00.000+02:00" is a value of type "xs:dateTime" representing nine o'clock in the evening of December

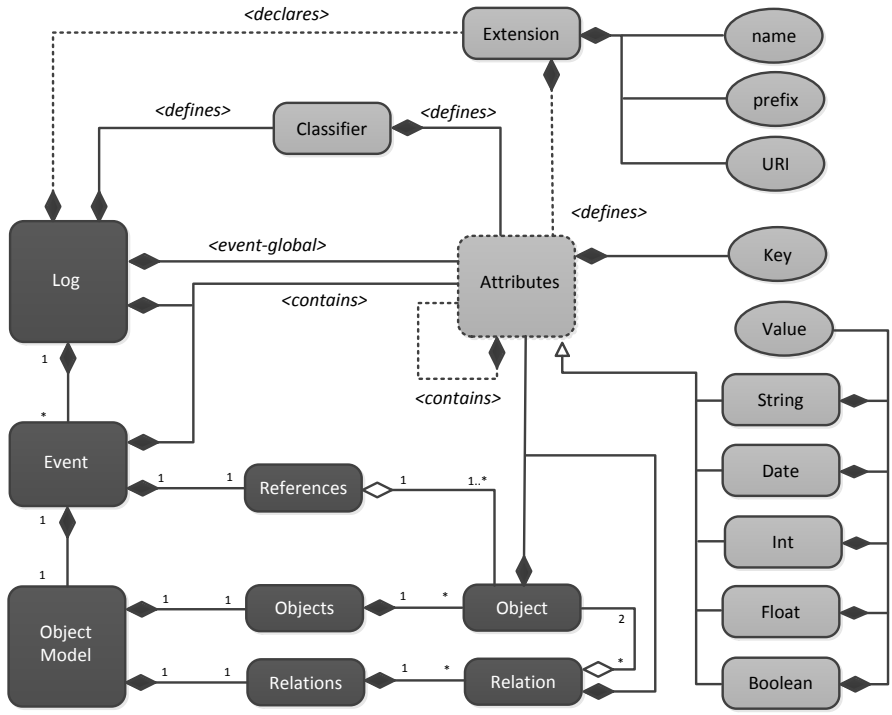


Figure 4.5: Meta model of XOC [136].

17th, 2011 in timezone GMT+2. For simplicity, we use "2011-12-17 21:00:00" as format to represent time when the timezone is not necessary.

- *Extension*. Similar to XES, XOC does not limit the attributes to a prescribed and fixed set. Users can declare their own attributes based on specific needs. In order to provide semantics for such attributes, the log refers to so-called *extensions*. An extension gives semantics to particular attributes. For example, the "Time" extension defines a timestamp attribute of type "xs:dateTime". The "Organizational" extension defines a resource attribute of type "xs:string". It is possible for users to develop domain-specific or even organization-specific extensions.

As shown in Figure 4.6, XOC declares particular attributes to be mandatory for different elements. For example, it may be stated that each event, object and

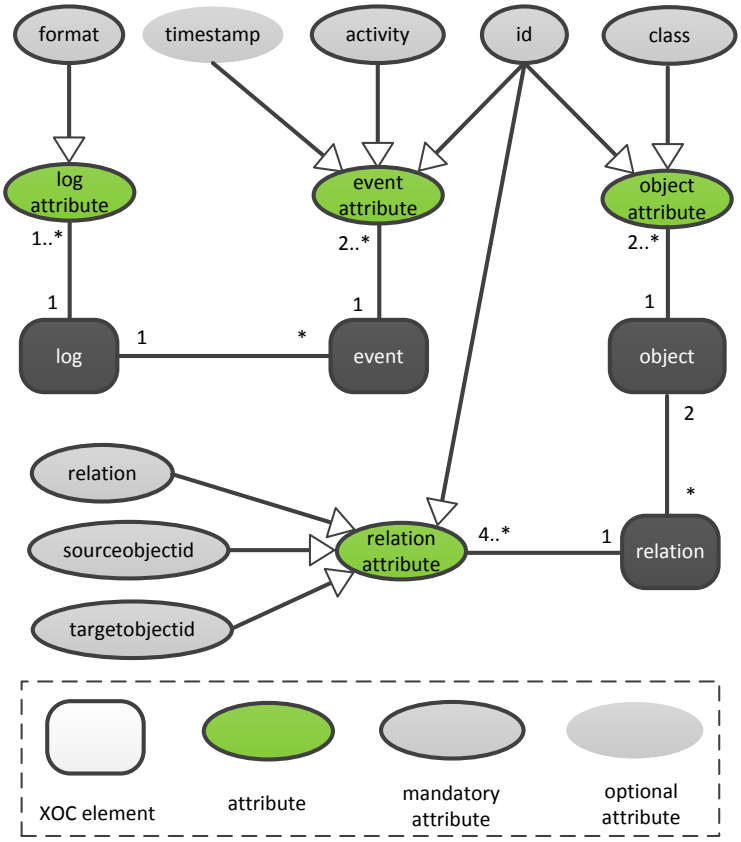


Figure 4.6: Zooming in on attributes of different elements.

object relation should have an attribute “id” for reference. Besides, each event must have a corresponding activity and each object must have a corresponding class. In addition to “id”, each object relation has three other mandatory attributes, in which the “relation” attribute indicates the relation type and the “sourceobjectid”/“targetobjectid” attribute indicates the source/target object. Note that the log has one mandatory attribute “format” which may have two possible values, “total” or “update”. It indicates the format of the log file, which is illustrated in Section 4.2.4. In addition to mandatory attributes, optional

attributes can be defined by users for each element. For instance, “timestamp” and “resource” can be optional attributes of events.

The XOC meta model shown in Figure 4.5 does not prescribe a concrete syntax and many serializations are possible. Therefore, a standard XML serialization is necessary to exchange XOC documents. Figure 4.7 shows a fragment of the XML serialization of the XOC log of Table 4.1. The tags of the XML serialization are:

- An <extension> tag declares an extension. For each extension, a shorter prefix is given, which is used in the attribute names. As shown in Figure 4.7, two extensions are declared: “Concept” and “Time”. Consider the “Time” extension as an example. It defines an attribute timestamp and uses prefix “time”. Accordingly, the timestamp of an event is stored using the key “time:timestamp”.
- A <global> tag defines the mandatory attributes for events, objects and object relations. Figure 4.6 only shows the smallest set of mandatory attributes and one can define more mandatory attributes. For instance, Figure 4.7 shows three global (mandatory) attributes for events (indicated by the “scope”), which means each event must contain these three attributes.
- A <classifier> tag defines the classifier function for the log. For instance, the “Activity” is defined as the classifier here.
- An <event> tag corresponds to an event, which consists of some attributes, an object model (i.e., <model> tag) and a set of object references (i.e., <references> tag).
- A <references> tag corresponds to a set of object references, in which each object (reference) (i.e., <object> tag) only contains the mandatory attribute (i.e., “id”) for simplicity.
- A <model> tag represents an object model which consists of a set of objects (<objects> tag) and a set of object relations (<relations> tag).
- An <object> or <relation> tag describes an object or an object relation, respectively. Its attributes are listed in the tag. For instance, the <object> tag in Figure 4.7 represents an object, which has two mandatory attributes (“identity:id” and “class”) and one optional attribute (“amount”).

Each attribute of an event, object or object relation has an attribute name (indicated by the “key” tag) and a corresponding value (indicated by the “value” tag). The “string” or “date” tag indicates the type of the attribute.

The XOC log example in Figure 4.7 only shows one event. Normally a log has many events. If a log file contains a large number of events, its size may be huge, i.e., the log size increases dramatically along with the number of events and objects. Object models may increase dramatically since an object model contains unchanged contents of its preceding object model and new contents added by

```
<?xml version=''1.0'' encoding=''utf-8''?>
    <extension name=''Concept'' prefix=''concept'' uri=''http://...''/>
    <extension name=''Identity'' prefix=''identity'' uri=''http://...''/>
    <extension name=''Time'' prefix=''time'' uri=''http://...''/>
    ...
    <global scope=''event''>
        <string key=''identity:id'' value=''id''/>
      <string key=''concept:name'' value=''name''/>
      <date key=''time:timestamp'' value=''2017-08-11 10:33:37''/>
    </global>
    ...
    <classifier name=''Activity'' keys=''concept:name''/>
    ...
    <string key=''format'' value=''total''/>
    <event>
        <string key=''identity:id'' value=''co1''/>
        <string key=''concept:name'' value=''create order''/>
        <string key=''time:timestamp'' value=''2017-08-11 10:33:37''/>
        <string key=''org:resource'' value=''Jack''/>
        <model>
            <objects>
                <object>
                    <string key=''identity:id'' value=''o1''/>
                    <string key=''class'' value=''order''/>
                    <string key=''amount'' value=''1100.5''/>
                </object>
                ...
            </objects>
            <relations>
                <relation>
                    <string key=''id'' value=''r10-o1-ol1''/>
                    <string key=''relation'' value=''r10''/>
                    <string key=''sourceobjectid'' value=''o1''/>
                    <string key=''targetobjectid'' value=''ol1''/>
                </relation>
                ...
            </relations>
        </model>
        <references>
            <object>
                <string key=''id'' value=''o1''/>
            </object>
            ...
        </references>
    </event>
    ...
</log>
```

Figure 4.7: A fragment of the XML serialization of the XOC log of Table 4.1.

its corresponding event, as shown in Figure 4.4. This problem can be solved by storing only update information, which will be explained in the next section.

4.2.4 Lightweight XOC Log Files

In order to compress the log size, we employ some techniques to deal with the growing rate of object models, resulting in lightweight logs. Clearly, it is sufficient to just store the object references and updates in the event log. This makes the storage format much better scalable. Lightweight logs still conform to the XOC meta model, but are much smaller because only changes are stored. The basic idea is that, on the one hand, we remove the redundant information (e.g., unchanged objects) to decrease the log size; and on the other hand we keep the necessary information to make the compressed logs (i.e., lightweight logs) as complete as the original ones, i.e., it is possible to recover original logs based on the compressed logs.

To distinguish the original log files and lightweight log files, we add an attribute "format" on the log level. This attribute indicates the format used by the log file and can have two possible values, i.e., "total" (indicating original logs) and "update" (indicating lightweight logs). The object model in the "total" log files represents the whole state of a system just after an event happens. In comparison, the object model in the "update" log files represents the updated part of the state of the system just after an event happens. Note that for an event in the "update" log files, it is the accumulation (integration) of its corresponding object model and all previous object models that represent the state of the system just after the event happens.

For instance, Figure 4.8 shows an XOC log file of the "total" format, indicated by the "format" attribute. As we can see in the second event "create_order2", its object model keeps two unchanged objects ("order1" and "order_line1") and an unchanged object relation "r10-o1-ol1" from the object model of its preceding event "create_order1" (the omitted attributes of these objects have the same values). The event "create_order2" only adds the object "order2" into the object model without changing anything else in the object model.

Figure 4.9 shows a lightweight log (i.e., the "update" format) corresponding to the log in Figure 4.8. In the lightweight log, we remove the unchanged objects ("order1" and "order_line1") and object relation ("o-fk_order-ol1") from the object model of the event "create_order2", and keep the added object ("order2") remained.

Note that it is possible that an object is removed from an object model. In this situation, a really removed object cannot be distinguished with a hidden object

```
<?xml version=''1.0'' encoding=''utf-8''?>
...
    <string key=''format'' value=''total''/>
    <event>
        <string key=''id'' value=''create_order1''/>
        ...
        <model>
            <objects>
                <object>
                    <string key=''id'' value=''order1''/>
                    ...
                </object>
                <object>
                    <string key=''id'' value=''order_line1''/>
                    ...
                </object>
            </objects>
            <relations>
                <relation>
                    <string key=''id'' value=''r10-o1-ol1''/>
                    ...
                </relation>
            </relations>
        </model>
        ...
    </event>
    <event>
        <string key=''id'' value=''create_order2''/>
        ...
        <model>
            <objects>
                <object>
                    <string key=''id'' value=''order1''/>
                    ...
                </object>
                <object>
                    <string key=''id'' value=''order_line1''/>
                    ...
                </object>
                <object>
                    <string key=''id'' value=''order2''/>
                    ...
                </object>
            </objects>
            <relations>
                <relation>
                    <string key=''id'' value=''r10-o1-ol1''/>
                    ...
                </relation>
            </relations>
        </model>
        ...
    </event>
    ...
</log>
```

Figure 4.8: An XOC log of the “total” format.

```
<?xml version=''1.0'' encoding=''utf-8''?>
...
    <string key=''format'' value=''update''/>
    <event>
        <string key=''id'' value=''create_order1''/>
        ...
        <model>
            <objects>
                <object>
                    <string key=''id'' value=''order1''/>
                    <string key=''lifecycle'' value=''add''/>
                    ...
                </object>
                <object>
                    <string key=''id'' value=''order_line1''/>
                    <string key=''lifecycle'' value=''add''/>
                    ...
                </object>
            </objects>
            <relations>
                <relation>
                    <string key=''id'' value=''o-fk_order-ol1''/>
                    <string key=''lifecycle'' value=''add''/>
                    ...
                </relation>
            </relations>
        </model>
        ...
    </event>
    <event>
        <string key=''id'' value=''create_order2''/>
        ...
        <model>
            <objects>
                <object>
                    <string key=''id'' value=''order2''/>
                    <string key=''lifecycle'' value=''add''/>
                    ...
                </object>
            </objects>
            <relations>
            </relations>
        </model>
        ...
    </event>
    ...
</log>
```

Figure 4.9: An XOC log of the “update” format.

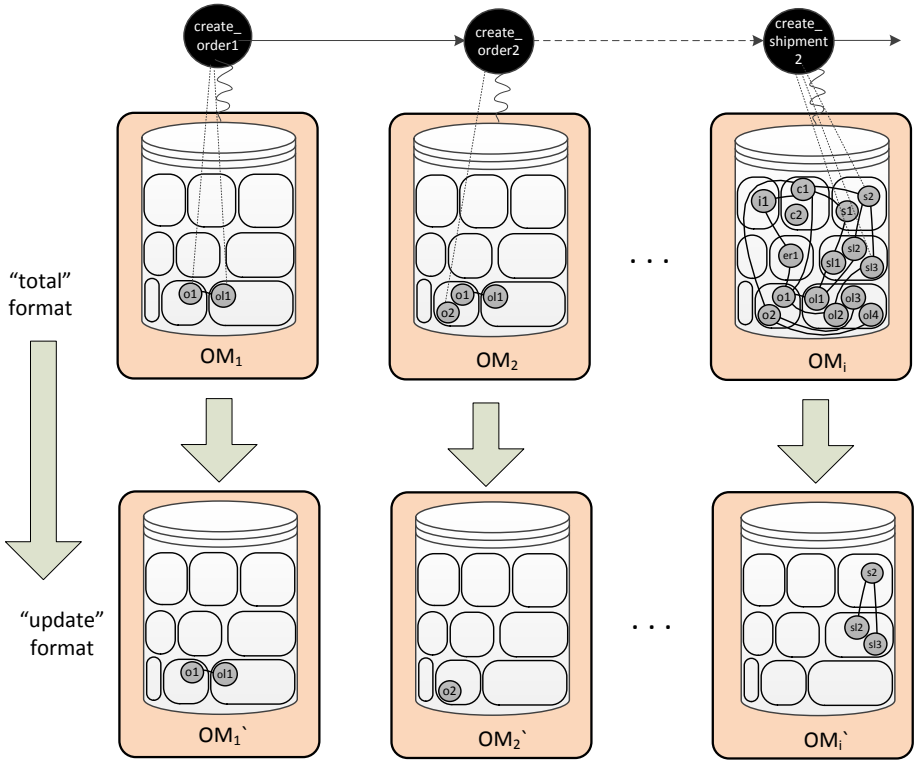


Figure 4.10: Transforming "total" object models into "update" object models.

(i.e., an unchanged object) in the lightweight logs. To solve this problem, we add an extra mandatory attribute "lifecycle" for each object and object relation in the lightweight log files. This attribute has three possible values, i.e., "add", "update", and "delete". It indicates the modification type of an object. By comparing two successive object models (we refer to the former object model as OM_1 and the latter object model as OM_2) of the "total" format, it is possible to infer OM_2' (the second object model of the "update" format). For that, the following rules are used:

- *Added objects and object relations*. If an object or object relation (i.e., its id) is only observed in OM_2, we conclude that the object or object relation is

added in OM_2. We copy it in OM_2' and add the attribute "lifecycle" with the "add" value on it.
- *Updated object and object relations*. If an object or object relation is observed in both object models, but with different attributes, we conclude that the object or object relation is updated in OM_2. We keep it in OM_2' and add the attribute "lifecycle" with the "update" value on it.
- *Unchanged object and object relations*. If an object or object relation is observed in both object models and has the same attributes, we conclude that the object or object relation is unchanged in OM_2. It is *not* included in OM_2'.
- *Removed objects and object relations*. If an object or object relation is only observed in OM_1, we conclude that the object or object relation is removed in OM_2. We copy it in OM_2' and add the attribute "lifecycle" with the "removed" value on it.

Note that if OM_2 is the first object model in the log (i.e., OM_1 does not exist), all object and object relations in OM_2 are considered as added objects and object relations. In other words, we copy all objects and object relations into OM_2', adding the attribute "lifecycle" with the "added" value on each object and object relation.

For instance, there exist two object models of the "total" format in Figure 4.8, i.e., OM_1 (corresponding to the event "create_order1") and OM_2 (corresponding to the event "create_order2"). Figure 4.9 shows two corresponding object models OM_1' and OM_2' (of the "update" format). Since OM_1 is the first object model in the log, its corresponding object model OM_1' has the same objects and object relations with an extra attribute "lifecycle" with the "added" value. For OM_2, its corresponding object model OM_2' only has one object "order2", since only "order2" is added in OM_2. Figure 4.10 presents the transformation mentioned above. Obviously, for a "total" object model, i.e., OM_i , its corresponding "update" object model, i.e., OM_i' takes less storage.

4.3 Extracting Logs from Object-Centric Event Data

In Chapter 3, we defined object-centric event data to abstract the data (e.g., database tables and other documents such as redo logs) generated by artifact-centric information systems. In this section, we propose an approach to extract XOC logs defined in Section 4.2 from object-centric event data, which enables further process mining techniques (which take logs as input) [91].

In XOC logs, each event has a corresponding object model and a corre-

sponding activity. Therefore, our approach basically consists of three steps: (i) recovering the object model for each event, (ii) identifying activities, and (iii) extracting XOC logs based on the first two steps.

4.3.1 Recovering Historical Object Models

An object-centric event data $OCED = (OM, ES, relate)$ set (cf. Chapter 3) has only one object model OM which presents the last state of the process after the effects of all previous events in ES. In other words, the object model corresponds to the last event in ES. In XOC logs, each event has a corresponding object model to show the state of the system just after the event. Therefore, we need to recover the historical (previous) object model to each event.

The idea to recover previous object models is based on two elements: a known object model and the effects of previous events on object models. For instance, if we know an object model OM corresponding to an event e and the effect of e on the object model, we can infer the object model preceding OM (i.e., the object model corresponding to the event preceding e) by removing (or inverting) the effect of e on OM.

An event corresponds to an operation on information systems, which modifies the state of the system by modifying the corresponding database. Each event can correspond to multiple database changes (an event consists of a sequence of changes, cf. Chapter 3). A database change corresponds to the execution of an SQL sentence, which is the smallest unit to modify the object model (representing the state of a database). Note that not every SQL sentence can be executed on the database successfully. For instance, it fails if we execute an SQL sentence to delete an inexistent record. In other words, only SQL sentences valid for the current state of a database can be executed successfully. Similarly, only changes valid for an object model can modify the object model successfully.

Definition 4.2 (Valid Changes) *Let $DM = (C, Attr, classAttr, val, PK, FK, classPK, classFK, keyRel, keyAttr, refAttr)$ be a data model and $OM = (Obj, Rel, class, attrO) \in VOM$ be a valid object model. A change $(op, c, Attr', map_{old}, map_{new}) \in CH^{DM}$ is valid for OM if and only if*

- $op = \oplus$, $\nexists (c', map) \in OM : c' = c \wedge \forall attr \in keyAttr(classPK(c)) : map(attr) = map_{new}(attr)$*, i.e., it is impossible to add an already existing object, or*
- $op \in \{\oslash, \ominus\}$, $\exists (c', map'_{old}) \in OM : c' = c \wedge map'_{old} = map_{old}$*, i.e., it is not possible to update or delete a non-existent object.*

A change is valid if it adds an inexistent object, i.e., which has different primary key values from any existing object, or it updates or deletes an existing

object in the object model. A valid change for an object model means that it is permissible in the object model. Otherwise, it will fail and the object model will remain unchanged.

Definition 4.3 (Effect of a Change Occurrence) *Let DM be a data model.* $OM = (Obj, Rel, class, attrO)$ *and* $OM' = (Obj', Rel', class', attrO')$ *are two valid object models.* $co = ((op, c, Attr', map_{old}, map_{new}), ts) \in CO^{DM}$ *is an occurrence of a valid change for OM. OM′ is generated after the change occurrence co on OM and denote* $OM \xrightarrow{co} OM'$ *or* $OM' \xleftarrow{co} OM$ *if and only if*

- $Obj' = \{o \in Obj \mid o \neq (c, map_{old})\} \cup \{(c, map_{new}) \mid op \neq \ominus\}$ *or*
- $Obj = \{(o \in Obj' \mid o \neq (c, map_{new})\} \cup \{(c, map_{old}) \mid op \neq \oplus\}$.

Given a valid object model OM, a change results in a new valid object model OM' by adding, updating or deleting an object in the object set Obj. The resulting object set Obj' contains all unchanged objects (i.e., $\{o \in Obj \mid o \neq (c, map_{old})\}$) and the added or updated new object (i.e., $\{(c, map_{new}) \mid op \neq \ominus\}$). Similarly, if given the resulting object model, we can also infer the object model before the change (indicated by the second rule in Definition 4.3).

Note that in a valid object model, the object relation set Rel and the functions $class$ and $attrO$ depend on the object set Obj. In other words, after deriving the object set Obj' based on Definition 4.3, we can specify Rel', $class'$ and $attrO'$ and then the complete object model OM'.

Definition 4.4 (Effect of Events) *Let DM be a data model and* $e = \langle co_1, co_2, \ldots, co_n \rangle \in E^{DM}$ *be an event. The event e results in a valid object model* OM_n *when starting in* OM_0*, denoted by* $OM_0 \xrightarrow{e} OM_n$ *or* $OM_n \xleftarrow{e} OM_0$*, if and only if there exist valid object models* $OM_0, OM_1, \ldots, OM_n \in VOM$ *such that* $OM_0 \xrightarrow{co_1} OM_1 \xrightarrow{co_2} OM_2 \ldots \xrightarrow{co_n} OM_n$.

Similarly, an event sequence $es = \langle e_1, e_2, \ldots, e_n \rangle \in (E^{DM})^*$ *results in a valid object model* OM_n *when starting in* OM_0*, denoted by* $OM_0 \xrightarrow{es} OM_n$ *or* $OM_n \xleftarrow{es} OM_0$*, if and only if there exist valid object models* $OM_0, OM_1, \ldots, OM_n \in VOM$ *such that* $OM_0 \xrightarrow{e_1} OM_1 \xrightarrow{e_2} OM_2 \ldots \xrightarrow{e_n} OM_n$.

Based on the effect of a change on an object model, an event results in an object model OM_n through orderly accumulating the effects of all changes of the event on the initial object model OM_0. Similarly, an event sequence results in an object model OM_n through orderly accumulating the effects of all events of the event sequence on the initial object model OM_0.

4.3.2 Identifying Activities

In an XOC log, each event has a corresponding activity to indicate the type of operations (e.g., "create order") on the information systems. The events in object-centric event data are sequences of database changes, e.g., $e = \langle((\oplus, o, Attr, map_{old}, map_{new}), ts), ((\oplus, ol, Attr', map'_{old}, map'_{new}), ts)\rangle$. They are raw (on the database level) and difficult to be understood by users. In this section, we identify activities (on the information system level) and build a mapping between events and activities to make the events more intuitive.

In order to connect events and activities, we define a notion of event type, which builds a bridge between events and activities. The idea is that each event corresponds to an event type and each event type has a corresponding activity (i.e., an activity can be considered as the name of an event type).

Definition 4.5 (Cardinalities) *$\mathcal{U}_{Card} = \mathcal{P}(\mathbb{N}) \setminus \{\emptyset\}$ defines the universe of all cardinalities.*

A cardinality (an element of $\mathcal{U}_{Card}$) specifies a non-empty set of integers. For instance, "1..*" is a cardinality which denotes any positive integers. Table 4.2 presents some frequently used cardinalities.

notation	allowed cardinalities
1	$\{1\}$
$1..k$	$\{1, 2, \ldots, k\}$
$*$	$\{0, 1, 2, \ldots\}$
$1..*$	$\{1, 2, \ldots\}$

Table 4.2: Some examples of frequently used shorthands for elements of $\mathcal{U}_{Card}$.

Definition 4.6 (Event Types and Activities) *Let DM be a data model and CT^{DM} be the set of change types of DM. $ET^{DM} = \{et \in \mathcal{P}(\mathcal{U}_{Card} \times CT^{DM}) \setminus \{\emptyset\} \mid \forall (card_1, ct_1), (card_2, ct_2) \in et : (ct_1 = ct_2 \Rightarrow card_1 = card_2)\}$ is the set of event types of DM.*

Let $ET \subseteq ET^{DM}$ be a set of event types specified based on domain knowledge. Function $extractA \in ET \rightarrow \mathcal{U}_A$ maps an event type to its name, i.e., its corresponding activity.

An *event type* is defined as a set of tuples of a *cardinality* and a change type, where the cardinality describes the quantitative relation between the event type

and the change type. One can use domain knowledge to specify event types. For instance, based on the knowledge that a click on the "create order" button inserts one record in the "order" table and one or more records in the "order_line" table, we can specify an event type named "create order", which consists of two change types, i.e., "insert one order record" and "insert one or more order line records". More precisely, the event type is denoted as $\{(\{1\}, (\oplus, o, Attr)), (\{1..*\}, (\oplus, ol, Attr'))\}$ which means a "create order" event adds ($\oplus$) precisely one ($\{1\}$) record (populating the attributes in $Attr$) in the "order" (o) table, and adds ($\oplus$) at least one ($\{1..*\}$) record (populating the attributes in $Attr'$) in the "order_line" (ol) table.

Based on domain knowledge, users can specify a set of event types and the corresponding activity for each type. In XOC logs, each event corresponds to precisely one activity. Accordingly, each event should also correspond to precisely one event type, since an activity can be considered as the name of an event type. Therefore, we should guarantee that the given set of event types cannot have multiple event types that can trigger the same event.

Definition 4.7 (Possible Events of an Event Type) *Let DM be a data model. Function $possibleE \in ET^{DM} \rightarrow \mathcal{P}(E^{DM})$ returns possible events of an event type such that $possibleE(et) = \{e \in E^{DM} \mid \forall((op, c, Attr, map_{old}, map_{new}), ts) \in e : (\exists card \in \mathcal{U}_{card} : (card, (op, c, Attr)) \in et) \wedge \forall(card', (op', c', Attr')) \in et : |\{((op', c', Attr', map'_{old}, map'_{new}), ts') \in e\}| \in card'\}$.*

For short, we denote $possibleE(ET) = \{e \mid \exists et \in ET : e \in possibleE(et)\}$ for a set of event types ET.

Each event type can trigger a set of possible events, specified by the function *possibleE*. An event is a possible event of an event type if (i) the types of all changes in the event are covered by the event type, and (ii) the cardinality of each change type contains the number of changes (in the event) of this type. For instance, an event with three changes $e = \langle((\oplus, o, Attr, map_{old}, map_{new}), ts), ((\oplus, ol, Attr', map'_{old}, map'_{new}), ts), ((\oplus, ol, Attr', map''_{old}, map''_{new}), ts)\rangle$ is a possible event of the "create order" event type $co = \{(\{1\}, (\oplus, o, Attr)), (\{1..*\}, (\oplus, ol, Attr'))\}$, because (i) the types of all changes in the event e are $\{(\oplus, o, Attr), (\oplus, ol, Attr')\}$, which can be found in co, and (ii) the number of changes of $(\oplus, o, Attr)$ and $(\oplus, ol, Attr')$ in e are 1 and 2, which are contained by the cardinalities $\{1\}$ and $\{1..*\}$, respectively.

Definition 4.8 (Valid Event Type Set) *Let DM be a data model. $ET \subseteq ET^{DM}$ is a valid event type set if and only if $\forall et, et' \in ET : possibleE(et) \cap possibleE(et') = \emptyset$.*

An event type set is valid if no event is a possible event of two different event types in the set. In other words, the possible events of any two event types in the

set are disjoint. In this way, we can guarantee that each event has at most one corresponding event type in the set.

Definition 4.9 (Extracting Event Types) *Let DM be a data model. $ET \subseteq ET^{DM}$ is a valid set of event types. $E \subseteq possibleE(ET)$ is a set of possible events of ET. Function $extractET \in E \rightarrow ET$ maps an event e to an event type et such that $extractET(e) = et$ where $e \in possibleE(et)$.*

Given a valid set of event types, the function *extractET* maps an event to an event type in the set. Up to now, we defined all the functions to map an event to an activity. For instance, the activity of an event e is $extractA(extractET(e))$. In the next section, we extract event logs based on these functions.

4.3.3 Extracting XOC Event Logs

The input for our approach (i.e., extracting XOC logs from artifact-centric information systems) is an object-centric event data set $OCED = (OM, ES, relate)$, which contains an object model OM, an event sequence ES and a function $relate$ relating events in ES to objects in OM. Section 4.3.1 showed how to create an object model for each event while Section 4.3.2 identified an activity for each event. This allows us to extract XOC event logs from object-centric event data.

Definition 4.10 (Extracting XOC Event Log) *Let DM be a data model and $OCED = (OM, ES, relate)$ be an OCED set. Function $extractA \in ET \rightarrow \mathcal{U}_A$ maps an event type from a predefined set to an activity based on domain knowledge. Function $extractL \in \mathcal{U}_{OCED} \rightarrow \mathcal{U}_L$ extracts an XOC event log from an OCED set such that $\forall OCED \in \mathcal{U}_{OCED} : extractL(OCED) = (E, act, attrE, relate, om, \preceq)$ where*

- *$E \subseteq E^{DM}$ is a set of events, where $E = \{e \in ES \mid \exists et \in ET : e \in possibleE(et)\}$,*
- *$act \in E \rightarrow \mathcal{U}_A$ maps events onto activities, such that $\forall e \in E : act(e) = extractA(extractET(e))$,*
- *$attrE \in E \rightarrow (\mathcal{U}_{Attr} \not\rightarrow \mathcal{U}_{Val})$ maps events onto a partial function assigning values to some attributes,*
- *$relate \in E \rightarrow \mathcal{P}(O^{DM})$ relates events to sets of objects, which is inherited from OCED.*
- *$om \in E \rightarrow \mathcal{U}_{OM}$ maps each event to an object model right after the event happened, such that $\forall e_i \in E : om(e_i) = OM_i$ where $OM \overset{ES'}{\leftarrow} OM_i$, $ES' = tl^{m-i}(\langle e_1, e_2, \ldots, e_m \rangle)$ and $\langle e_1, e_2, \ldots, e_m \rangle = ES$,*[2]

[2] $tl^k(\sigma)$ means to get the last k elements of a sequence σ.

- $\preceq \subseteq E \times E$ *defines a total order on events, where* $\preceq = \{(e_i, e_j) \mid e_i \in E \wedge e_j \in E \wedge e_j \in tl^{m-i+1}(\langle e_1, e_2, ..., e_m \rangle)\}$ *and* $\langle e_1, e_2, ..., e_m \rangle = ES$.

Given a set of event types and a corresponding activity for each type (which can be derived based on domain knowledge), function *extractL* extracts an XOC event log $L = (E, act, attrE, relate, om, \preceq)$ from an OCED set $OCED = (OM, ES, relate)$. Note that the event set E in L only contains the events in ES that are possible events of an event type in the given set. This enables users to filter events based on their needs. For instance, one can derive a customized event set E by giving a specific set of event types.

Function *act* identifies an activity for an event, which is the integration of *extractET* and *extractA*. More precisely, *extractET* maps an event onto an event type and *extractA* maps the event type onto an activity. Events can have attributes, indicated by *attrE*. For instance, the timestamp of an event can be one of its attributes, e.g., for $e = \langle((\oplus, o, Attr, map_{old}, map_{new}), ts), ((\oplus, ol, Attr', map'_{old}, map'_{new}), ts)\rangle$, $attrE(e)(timestamp) = ts$. Currently, we can only automatically derive the *timestamp* attribute for events. However, it is possible to add more attributes such as *resource* based on more available information (e.g., change tables record information about the users who make database changes). Function *relate* in L is the same as the function *relate* in *OCED*.

Function *om* is used to derive the object model for each event. The idea is that, starting from the object model OM in $OCED$ corresponding to the last event e_m in ES, we reverse the effects of event sequences on OM to get the previous object models. More precisely, in order to obtain the object model corresponding to one event e_i, we get (i) its suffix event sequence $ES' = tl^{m-i}(\langle e_1, e_2, ..., e_m \rangle)$ (i.e., $\langle e_{i+1}, e_{i+2}, ..., e_m \rangle$), and (ii) the object model OM_i through $OM \overset{ES'}{\leftarrow} OM_i$.

$\preceq$ defines a total order on events. It is a set of event pairs. Each event pair $(e_i, e_j) \in \preceq$ indicates the order of these two events e_i and e_j (consistent with the order in the event sequence). (e_i, e_j) indicates $i \leqslant j$, i.e., e_i happens before or at the same time of e_j. It can also be denoted as $e_i \preceq e_j$.

4.4 Evaluation

Section 4.2 defined XOC event logs while Section 4.3 proposed an approach to extract such logs from artifact-centric information systems. In this section, we evaluate the XOC format by comparing with the XES format from different angles.

4.4.1 Log Generation

Following the approach in Section 4.3, we can extract XOC logs from the database tables. There are also approaches to extract XES logs from database tables. In order to make a fair comparison between XOC logs and XES logs, we use the same data set for the generation of an XES log and an XOC log.

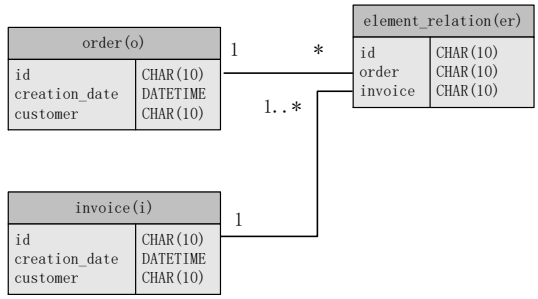


Figure 4.11: Class diagram of a database that consists of three tables.

Figure 4.11 shows a class diagram describing a database consisting of three tables: “order”, “element_relation” and “invoice”. These three tables consist of the “order-to-invoice” process from the order-to-cash scenario of a real ERP system named Dolibarr. For simplicity, we only include the important columns for each table.

id	creation_date	customer	amount
o1	2017-08-11 10:33:37	c1	3808
o2	2017-08-13 16:28:15	c1	1907

Table 4.3: Subset of records in the “order” table.

The “order” table contains information about orders. Table 4.3 shows the information of two orders. Each order has a unique id, refers to a customer and has a “create_date” attribute to indicate the creation time and an “amount” attribute to indicate the amount of the order. For instance, the order $o1$ is created at “2017-08-11 10:33:37” for customer $c1$ with an amount of 3808.

The “invoice” table stores all invoice information for orders. It has the same attributes as the “order” table. For instance, Table 4.4 shows three invoice

id	creation_date	customer	amount
i1	2017-08-15 09:13:27	c1	2380
i2	2017-08-17 17:38:36	c1	1431
i3	2017-08-23 14:23:19	c1	1904

Table 4.4: Subset of records in the "invoice" table.

records, in which the invoice $i1$ is created at "2017-08-15 09:13:27" for customer $c1$ with an amount of 2380.

id	order	invoice	amount
er1	o1	i1	2380
er2	o1	i2	1428
er3	o2	i2	3
er4	o2	i3	1904

Table 4.5: Subset of records in the "element_relation" table.

The "element_relation" table indicates the correspondence between orders and invoices. Each record has a unique id and refers to both an order and an invoice. Note that there may exist many-to-many relations between orders and invoices. As shown in Table 4.5, $er1$ and $er2$ indicate that one order $o1$ corresponds to two invoices $i1$ and $i2$ while $er2$ and $er3$ indicate that one invoice $i2$ corresponds to two orders $o1$ and $o2$.

activity	table	timestamp column
create order	order	creation_date
create invoice	invoice	creation_date

Table 4.6: Timestamp columns in tables corresponding to activities.

Clearly the values in timestamp columns in "order" and "invoice" tables can be considered as events related to the "order-to-invoice" process. Table 4.6 shows the activities and corresponding timestamp columns. For instance, we map the "creation_date" column in the "order" table to the "create order" activity, which means each value in "creation_date" is the timestamp of a "create order" event.

case id	activity	timestamp	other attributes
o1	create order	2017-08-11 10:33:37	customer:c1, amount:3808
o1	create invoice	2017-08-15 09:13:27	customer:c1, amount:2380
o1	create invoice	2017-08-17 17:38:36	customer:c1, amount:1431
o2	create order	2017-08-13 16:28:15	customer:c1, amount:1907
o2	create invoice	2017-08-17 17:38:36	customer:c1, amount:1431
o2	create invoice	2017-08-23 14:23:19	customer:c1, amount:1904

Table 4.7: An XES log of events extracted from all three tables using "order" as the case notion.

When creating an XES event log, each event needs to be associated with a particular case. Therefore, we need to flatten the three tables into one table with a "case id" column. However, one can choose from three types of cases: orders, element relations, invoices. Any record in one of the three tables potentially corresponds to a case. Assume that we are mainly interested in orders. Therefore, each case corresponds to a record in the "order" table. Table 4.7 shows an XES log of events extracted from all three tables using "order" as the case notion. It flattens the original database consisting of three tables into one table with a "case id" column. More precisely, since there are two records in the "order" table, the derived XES log consists of two cases, i.e., $o1$ and $o2$. The flattened event log is like a view on the complete data set. It is possible to use another way (e.g., considering an invoice as a case) to flatten the original database, i.e., alternative views are possible based on users' needs.

Based on the three tables and some extra information, such as change tables or domain knowledge, we can extract an XOC log as shown in Table 4.8. The XOC log has 5 events, i.e., two "create order" events and three "create invoice" events. Each row indicates the activity, attributes, object references and object model for an event. For instance, the fourth event is a "create invoice" event; there is one timestamp attribute; it refers to objects $i2$, $er2$ and $er3$; it corresponds to an object model consisting of 7 objects ($o1$, $o2$, $i1$, $er1$, $i2$, $er2$ and $er3$) and 6 object relations $((r3, i1, er1)$, $(r8, o1, er1)$, $(r3, i2, er2)$, $(r8, o1, er2)$, $(r3, i2, er3)$ and $(r8, o2, er3))$. Note that we present some contents for each object in the bracket after the object. For instance, $o1(id : o1, class : order, customer : c1, amount : 3808)$ means the class of $o1$ is "order" and it has two attributes $customer : c1$ and $amount : 3808$. All events in the XOC logs are only adding objects in the object model, i.e., the contents of each object do not change in the log. Therefore, we only explicitly show the contents of added objects for each event.

activity	attributes	references	object model	
			objects	relations
create order	timestamp:2017-08-11 10:33:37	o1	o1(id:o1,class:order,customer:c1,amount:3808)	
create order	timestamp:2017-08-13 16:28:15	o2	o1,o2(id:o2,class:order,customer:c1,amount:1907)	
create invoice	timestamp:2017-08-15 09:13:27	i1,er1	o1,o2,i1(id:i1,class:invoice,customer:c1,amount:2380), er1(id:er1,class:element_relation,customer:c1,amount:2380)	(r3,i1,er1), (r8,o1,er1)
create invoice	timestamp:2017-08-17 17:38:36	i2,er2,er3	o1,o2,i1,er1,i2(id:i2,class:invoice,customer:c1,amount:1431), er2(id:er2,class:element_relation,customer:c1,amount:1428), er3(id:er3,class:element_relation,customer:c1,amount:3)	(r3,i1,er1), (r8,o1,er1), (r3,i2,er2), (r8,o1,er2), (r3,i2,er3), (r8,o2,er3)
create invoice	timestamp:2017-08-23 14:23:19	i3,er4	o1,o2,i1,er1,i2,er2,er3, i3(id:i3,class:invoice,customer:c1,amount:1904), er4(id:er4,class:element_relation,customer:c1,amount:1904)	(r3,i1,er1), (r8,o1,er1), (r3,i2,er2), (r8,o1,er2), (r3,i2,er3), (r8,o2,er3), (r3,i3,er4), (r8,o2,er4)

Table 4.8: An XOC log of events extracted from all three tables without the case notion.

4.4.2 Log Structure

If we compare the generated XES log in Table 4.7 and XOC log in Table 4.8, it is noticeable that the two logs organize information from the three tables in different ways. In general, XES logs organize information in an event-centric way while XOC logs organize information in an object-centric way.

In the XES log in Table 4.7, all information is structured based on events, i.e., events are first identified based on timestamp values and then the information related to events is transformed into the activity, timestamp and other attributes attached to each event. However, the event-centric principle is not consistent with the features of the data from database tables, which are essentially object-centric. In databases, all information related to an object is gathered as a record in a table. A record can contain none to multiple events since a record can contain any number of timestamp values. If we want to include all information from the database tables, we have to relate each value in a record to an event and transform the value as an attribute of the event. In this sense, the information not directly related to events is discarded. For instance, since there are no events (i.e., timestamp values) in the "element_relation" table, the XES log does not explicitly contain any information from the table (some information is implicitly contained since they are used to relate "create invoice" to "create order" events). A value in the "amount" column indicates how much an invoice pays for an order, which is quite useful but cannot be reflected any more.

In comparison, the XOC log in Table 4.8 is object-centric, which is more suitable to deal with the data from database tables. In the transformation process, each timestamp value in a record is considered as an extracted event and the information related to the event is considered as the attributes of the event. No matter if a record has events, all information (it is possible to only contain useful information) in the record is abstracted as an object of a class corresponding to the table. For instance, all records in the "element_relation" table are extracted as objects ($er1$, $er2$, $er3$ and $er4$) in the XOC log (although there are no events in the table), which retains the information to indicate the correspondence between invoices and orders in terms of *amount*. In summary, XOC logs can contain all information from database tables (i.e., it is possible to restore the database tables based on XOC logs) while XES may lose important information which are not related to events. Note that XOC logs can keep information from databases as complete as they are, which does not mean that XOC logs should always contain all information of a database. In real applications, a database usually involves multiple business processes and an XOC log often only contains the information related to the target process.

4.4.3 Log View

In order to enable process mining, events need to be related. Therefore, event logs need to provide a mechanism to relate events. Note that the mechanism depends on the contents and structure of event logs, which decides the view of process used by a process mining application, such as discovery, conformance checking and performance analysis.

In XES logs, the case notion is employed to related cases. The case notion is available based on the assumption that the process models behind the information systems are *flat* models, which describes the life-cycle of a case of a particular type (e.g., a compensation request). All activities in a flat process model correspond to status changes of such a case (e.g., request submitted, request checked and request approved). However, it is important to realize that real-life processes are *not flat*. For instance, the "order-to-cash" process on ERP systems involve multiple modules, such as order, invoice and delivery, where there exist complex many-to-many interactions between different modules.

An XES log provides a specific view on the complete data set based on a chosen case notion. For instance, the XES log in Table 4.7 shows the process from the view of "order". It flattens the database tables in an unrecoverable manner. In other words, the process of relating events based on the chosen case notion finishes when the XES log is generated (i.e., the event correlation is hard coded). It is impossible to restore the original data and relate events using another case notion based on the flat XES log. For instance, Table 4.7 shows an XES log which relates events by choosing "order" as the case notion. Considering only the log, we cannot restore the original "order", "element_relation" and "invoice" tables and relate events based on another case notion.

An XOC log does not assume a case notion to relate events. As shown in Table 4.8, the XOC log does not have a "case id" column, which is mandatory for an XES log in Table 4.7. In contrary, we use the "reference" and "object model" columns to relate events. More precisely, each event refers to some objects in the object model (indicated by the "reference" column), and objects are connected by object relations (indicated by the "relations" column). The objects can be considered as a bridge to relate events. For instance, if two events refer to the same object, they are related (cf. Chapter 6). In summary, XOC logs have the potential to relate events in a more flexible (customized) manner and provide multiple views of the process, while XES logs relate events in a hard code manner and provide a specific flat view.

4.4.4 Log Quality

Event logs are the starting point for process mining techniques. From a practical point of view, the quality of event logs is of the utmost importance for the success of process mining. If event data are missing or cannot be trusted, the results derived by process mining techniques are less valuable.

In this section, the XOC and XES logs used for comparison have more activities, i.e., “create order” (co), “create shipment” (cs), “create invoice” (ci) and “create payment” (cp). They are extracted from the “order”, “invoice”, “element_relation” tables and some other tables, using the same method described in Section 4.4.1. The XOC log contains two perspectives, i.e., the behavioral perspective (i.e., events) and the data perspective (i.e., object models) and the events can be correlated based on the objects in the object models. In contrast, the XES log only contains the behavioral perspective, and events from the behavioral perspective are correlated by case ids. In order to make a fair comparison, we correlate the events in the XOC log and only compare its correlated events with the cases in the XES log. In other words, the comparison in mainly on the behavioral perspective.

Figure 4.12 shows the correlated events (cf. Chapter 6 for details of correlating events) of the XOC log, i.e., $co1$, $co2$, $cs1$, $ci1$, $cs2$, $ci2$, $cs3$, $cp1$, $ci3$, $cp2$, which are ordered by time. The different colors indicate the events are of different activities, e.g., the events in red are of the “create order” (co) activity. Due to many-to-many relations between orders and invoices (indicated by Table 4.5), the XOC log has corresponding many-to-many relations between “create order” and “create invoice” events. For instance, the “create order” event $co1$ is related to two “create invoice” events $ci1$ and $ci2$ and the “create invoice” event $ci2$ is related to two “create order” events $co1$ and $co2$. As shown in Figure 4.12, it is clear that the XOC log is able to represent one-to-many and many-to-many relations.

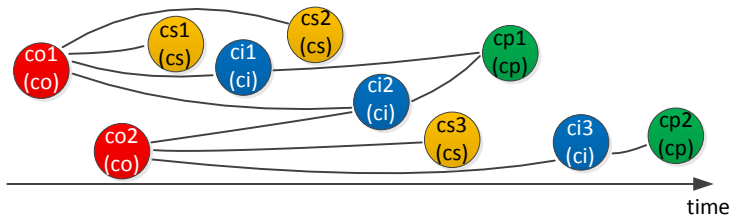


Figure 4.12: The behavioral perspective of the generated XOC log.

In contrast, Figure 4.13 presents two cases $o1$ and $o2$ of the XES log. The log is generated by flattening and splitting events based on a straightjacket case notion, i.e., "order". Due to one-to-many and many-to-many relations, the generation of the XES log suffers two well-known problems: data divergence and data convergence. Data convergence corresponds to the situation that one event refers to multiple cases. For instance, the event $ci2$ is related to two orders created by $co1$ and $co2$. Considering "order" as the case notion, $ci2$ is duplicated and split into two cases $o1$ and $o2$. The data convergence harms the log quality by leading to wrong frequencies of events because of event duplication. Data divergence means that a case is referred to by multiple events of the same activity. For instance, there are two events ($ci2$ and $ci3$) of the "create invoice" activity and two events ($cp1$ and $cp2$) of the "create payment" activity in the case $o2$. Due to the multiple instances of the same activity, the correspondences between different instances are misleading. For instance, in the case $o2$ we can infer that $cp1$ is related to $ci2$ and $cp2$ is related to $ci3$. However, it is not clear if $cp2$ is related to $ci2$, as it is possible that $cp1$ pays for a part of $ci2$ and $cp2$ pays for the other part of $ci2$ and the whole $ci3$.

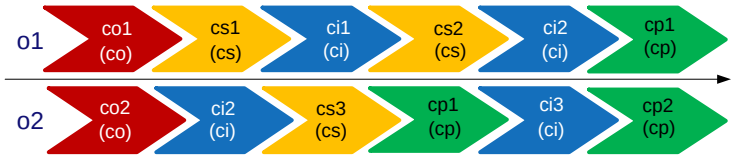


Figure 4.13: The generated XES log with data convergence ($ci2$ is related to two cases $o1$ and $o2$) and divergence ($o2$ has multiple instances of ci, i.e., $ci1$ and $ci2$).

In summary, by removing the case notion, the XOC log can present one-to-many and many-to-many relations as they are, and avoid the data convergence and data divergence problems. By improving the log quality, the results derived by later techniques are more reliable.

4.5 Related Work

Event logs serve as the input for many process mining techniques. In this section, we review the existing event log formats and introduce some researches which extract event logs from databases.

4.5.1 Event Log Formats

An event log is a predefined structure for storing event data. XES is the de facto log format for process mining. It stores information regarding the event log as a whole, the traces and the events belonging to the traces. It is supported by tools such as ProM and implemented by OpenXES, an open source java library for reading, storing and writing XES logs.

The XES format cannot deal well with object-centic data (e.g., database tables). González López de Murillas et al. [50] propose a meta model to abstract the object-centric data and redo logs into different types of entities such as attributes, objects, relations and events. The meta model covers both behavioral and data perspectives and builds a bridge to connect databases with process mining. This meta model provides all elements required in object-centric event data. In other words, the elements involved in object-centric event data, e.g., events and objects, are a subset of this meta model, but organized in a different way. This meta model is not a concrete log format, but transforms the object-centric data into "bricks" which can constitute logs to enable process mining techniques.

Berti [13] introduces a so-called *StarStar model*. It provides a multigraph visualization of the relationships between activities inferred from database events, connected with edges that are annotated with frequency and performance information. This model offers a simple way to identify a case notion to retrieve a classic event log.

Our XOC log format combines the "bricks" from [50] in an object-centric way, i.e., it focuses more on the data perspective, unlike the XES format. Objects replace a case notion to correlate events, which makes XOC logs able to deal with one-to-many and many-to-many relations. Moreover, by defining object models, an XOC log reveals the evolution of a database through time.

4.5.2 Extracting Event Logs

In order to obtain event logs, researchers have proposed a number of techniques and tools. [57] presented the ProM Import Framework (ProMimport), which was the first tool to extract MXML event logs. A currently popular tool is XESame[3], which is used to convert databases into XES event logs [157]. van der Aalst [135] conceptualizes a database view over event data based on the idea that events leave footprints by changing the underlying database, which are captured by

[3] http://www.processmining.org/xesame/start

redo logs of database systems. Triggered by this idea, González López de Murillas et al. [51] propose a method to extract XES logs from databases by identifying a specific trace id pattern. Calvanese et al. propose an approach [21] and a tool (*onprom*) [20] to flexibly extract XES event logs from relational databases, by leveraging the ontology-based data access paradigm.

In order to obtain logs from non-process-aware information systems, Pérez-Castillo et al. [111] correlate events into process instances using similarity of their attributes. Jans and Soffer [70] provide an overview of the decisions that impact the quality of the event logs constructed from database data. Ingvaldsen and Gulla [68] propose an analysis system which allows users to define events, resources and their inter-relations to extract logs from SAP transactions. Raichelson and Soffer [118] address merging logs produced by disintegrated systems that cover parts of the same process by choosing a "main" process. Artifact-centric approaches [93, 107] try to extract artifacts (i.e., business entities) and address the possibility of many-to-many relationships between artifacts.

Compared with the existing approaches, our approach outputs object-centric logs, i.e., XOC logs, rather than case-centric logs, i.e., XES logs. The main advantage of object-centric logs is the new kinds of analyses that they enable. Data-aware process model discovery [87], and new conformance checking techniques [141] that exploit data relations are examples of approaches directly applicable to XOC logs. Another important contribution is that our approach supports the abstraction from low-level database events (like the ones obtained in [51]) to high-level events (e.g., from the original information system).

4.6 Summary

In this chapter, we proposed a novel log format, i.e., the eXtensible Object-Centric (XOC) log format and an approach to extract event logs of such format from object-centric event data (i.e., database tables). The XOC format provides a process mining view on databases. Compared with existing log formats, such as XES, it has the following advantages:

- By removing the case notion and correlating events with objects, XOC logs can easily deal with one-to-many and many-to-many relations, avoiding convergence and divergence problems and displaying interactions between different instances.
- An object in XOC logs contains as much information as its corresponding record in the database. By extending the data perspective, XOC logs retain the data quality.

- The object model of an event represents a snapshot of the database just after the event occurrence. Based on this idea, the log provides a view of the evolution of the database, along with the operations which triggered changes in the database.

Besides, XOC logs serve as a starting point for a new line of analysis techniques. Based on experiments implemented on the generated XOC logs, it is possible to discover *Object-Centric Behavioral Constraint* (OCBC) models to describe the underlying process in an object-centric manner [87] (cf. Chapter 6). Additionally, taking an XOC log and an OCBC model as input, many deviations which cannot be detected by existing approaches, can be revealed by new conformance checking techniques [141] (cf. Chapter 7). Moreover, based on an XOC log and an OCBC model, performance analysis is also enabled to detect bottlenecks of business processes. Note that the OCBC discovery, conformance checking and performance analysis are just examples of potential applications of XOC logs. It is possible to develop more techniques based on XOC logs, e.g., discovering other types of data-aware process models.

Chapter 5

Object-Centric Behavioral Constraint Modeling Language

The first type of process mining is model discovery, i.e., discovering a process model from an event log to reveal the business process executed in real transactions. Chapter 2 analyzed the data generated by artifact-centric information systems and Chapter 3 defined the XOC event log format to organize these data for process mining techniques such as model discovery. Since the business processes on artifact-centric systems are different from the ones on WFM/BPM systems, conventional modeling languages fail to deal with these processes properly. In this chapter, we propose a novel modeling language, i.e., *Object-Centric Behavioral Constraint* (OCBC), to describe the processes on artifact-centric information systems, which can be discovered from XOC logs (cf. Chapter 6).

This chapter is organized as follows. In Section 5.1, we motivate OCBC models by discussing the challenges suffered by conventional models in terms of business processes on artifact-centric information systems. An OCBC model covers two perspectives of a business process, i.e., the data perspective and behavioral perspective, which are illustrated in Section 5.2 and Section 5.3, respectively. Section 5.4 defines OCBC models by integrating these two perspectives, i.e., adding the interactions in between. We evaluate OCBC models by comparing them with other models (such as Workflow nets and BPMN diagrams) in terms of describing business processes in Section 5.5. Section 5.6 reviews the related work and Section 5.7 concludes this chapter.

5.1 Motivation for Object-Centric Behavioral Constraint Models

Business process modeling languages play a significant role in describing and understanding business processes from different perspectives, such as the behavioral perspective (i.e., control-flow), data perspective and resource perspective. State-of-the-art process modeling languages (BPMN diagrams [55], Petri nets [148], EPCs [132], UML activity diagrams [16]) often assume a case notion in business processes, i.e., each event is related to exactly one case identifier and one process instance is recorded as a set of events which have the same case identifier.

However, when it comes to artifact-centric/data-centric processes supported by CRM and ERP systems, most of the existing languages fail. More precisely, they often force the analyst or modeler to straightjacket real-life processes from artifact-centric systems into flat or separate models that fail to capture the essential features. The typical reason for the failure is due to a missing case notion in these processes. Moreover, artifact-centric systems (e.g., CRM and ERP) support business functions related to sales, procurement, production, accounting, etc. These systems may contain hundreds, if not thousands, of tables with information about customers, orders, deliveries, etc. There exist one-to-many and many-to-many relationships between entities such that it is impossible to identify a global and unique process instance notion for all the entities, i.e., a clear case notion is missing in such systems.

Besides, since an artifact-centric information system consists of interactive modules (corresponding to different entities), the data perspective which presents the structure (i.e., mutual relationships among entities) is key to the system. In contrast, the behavioral perspective of the system is only an implicit and loose business process (since the system has to deal with flexible business processes), which can be considered as a second-class citizen. Apparently, the essence of the artifact-centric information systems conflicts with existing modeling languages, which often focus on the behavioral perspective, i.e., they only describe the lifecycles of individual instances (cases) in isolation. Although process models may include data elements (cf. BPMN [48]), they cannot provide a strong data perspective to describe the artifact-centric systems, since explicit connections to *real* data models (e.g., an ER model [25] or a UML class model) are rarely made.

The mismatch discussed above between process modeling languages and the actual processes supported by artifact-centric systems becomes clear when applying process mining to data from such systems. In summary, conventional

business process modeling languages tend to suffer the following problems:

- It is *difficult to identify a case notion* for the whole process. Due to one-to-many and many-to-many relationships between entities in business processes, a global and unique case notion is missing. For instance, there exists a many-to-many relationship between the "sales" and "delivery" departments. In the eyes of the first department, the case notion is "order", whereas in the second one, the case notion is "shipment". Therefore, it is difficult to find a common case notion for these two departments.
- It is *difficult to model interactions between process instances*. Mainstream business process modeling notations can only describe the lifecycle of one type of process instance at a time, such that the interactions between instances are missing. Consider for example BPMN. Although BPMN uses concepts like lanes, pools, and message flows to address this problem, a single instance is still modeled in isolation within each (sub)process.
- The *data perspective is underestimated*. Data elements can be modeled, but the more powerful constructs present in ER models and UML class models cannot be expressed well in conventional process models.
- It is *difficult to model an end-to-end process in an integrated diagram*. Since there are multiple case notions, the whole process is often distributed over multiple diagrams. Besides, a good combination of the data perspective and behavioral (control-flow) perspective is not reflected in today's process models. For instance, cardinality constraints in the data model must influence behavior, but this is missing in models such as BPMN diagrams and data-aware Petri nets.
- *A unified manner is not employed* for constraints on different perspectives. Constraints on different perspectives influence each other, e.g., cardinality constraints in the data model *must* influence behavior. Therefore, a consistent constraint manner is needed to precisely describe interactions between different perspectives.

The problems mentioned above have been around for quite some time (see for example [154]), but were never solved satisfactorily. Artifact-centric approaches [28, 66, 92, 107] (including the earlier work on proclets [140]) are state-of-the-art approaches to deal with one-to-many and many-to-many relationships and integrate data and behavioral perspectives. However, they still force one to pick an instance notion for each artifact, although a case notion for the whole process is not required. Moreover, the control-flow cannot be related to an overall data model (i.e., there is no explicit data model or it is separated from the control-flow) and interactions between different entities are not visible (because artifacts are distributed over multiple diagrams). For other conventional

languages like BPMN, concepts like lanes, pools, and message flows are used to describe the interactions between instances of (sub)processes. However, within each (sub)process a single instance is still modeled in isolation. Colored Petri nets (CPNs) [73] and data-aware Petri nets (DPNs) [33, 127] try to add a data perspective to Petri nets, e.g., a token can have attributes and activities can write/read data elements (variables and objects). However, the more powerful notations used in data modeling language (e.g., a UML class model) are rarely employed. In contrast, temporal data models try to add a behavioral perspective by describing how data objects evolve over time, but they do not model activities explicitly [9, 52].

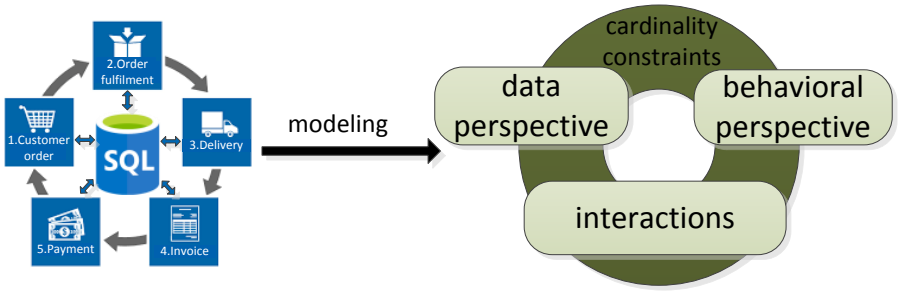


Figure 5.1: The idea of Object-Centric Behavioral Constraint (OCBC) models: describing the data perspective and behavioral perspective of business processes as well as their interactions in one single diagram with a unifying mechanism, i.e., cardinality constraints.

In order to totally (or better) solve these problems, this chapter proposes a novel modeling language, i.e., the *Object-Centric Behavioral Constraint* (OCBC) modeling language [90, 138, 141]. It combines ideas from declarative, constraint-based languages like *Declare* [113, 143, 143], and data/object modeling techniques like UML. An OCBC model covers both data and behavioral perspectives and the interactions in between. Cardinality constraints are used as a *unifying mechanism* to tackle data and behavioral dependencies, as well as their interplay(cf. Figure 5.1). Essentially, an OCBC model *extends a class model with a behavioral perspective*. More precisely, class models (referring to UML class models) are used to describe the data perspective of business processes since they can easily deal with one-to-many and many-to-many relationships. This ability is exploited to create process models that can also describe complex interactions (e.g., many-to-many relationships) between different types of instances.

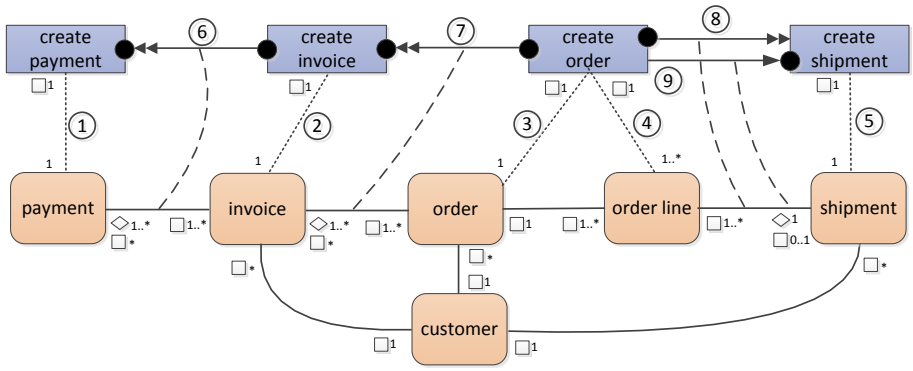


Figure 5.2: An *Object-Centric Behavioral Constraint* (OCBC) model, which describes the OTC process.

Classical multiple-instance problems are handled by using the data model for event correlation. In terms of the behavioral perspective (referring to Declare models), the declarative nature of the proposed language makes it possible to model behavioral constraints over activities like cardinality constraints in data models. In summary, the resulting OCBC model is able to describe processes involving *interacting instances* and *complex data dependencies*.

Figure 5.2 shows an OCBC model example to describe the Order To Cash (OTC) process, which is one of the most typical business processes supported by ERP systems. The OTC process has many variants and this example is a variant employed by *Dolibarr*. In general, an OCBC model consists of three parts, which are explained as follows. The top part shows behavioral constraints (i.e., the behavioral perspective of the process). Those constraints describe the ordering of activities (*create order*, *create invoice*, *create payment*, and *create shipment*). The bottom part describes the structure of entities relevant for the process (i.e., the data perspective), which can be read as if it was a UML class model (with six object classes *order*, *order line*, *invoice*, *payment*, *shipment*, and *customer*). The middle part integrates these two perspectives by relating activities, constraints, and classes. The notation of OCBC models will be explained in more detail later.

Obviously, the process described in Figure 5.2 has one-to-many and many-to-many relationships, and it is impossible to identify a single case notion. For instance, *one payment* can cover *multiple invoices* and *multiple payments* can be executed for *a particular invoice* (i.e., one payment only covers a part of the

invoice). Besides, different types of instances are intertwined and constraints in the class model influence the allowed behavior in the process. Therefore, the process cannot be modeled using conventional notations (e.g., BPMN).

5.2 Modeling the Data Perspective

The data perspective of a business process describes the entities and relationships between entities involved in the process. For example, the OTC business process has entities, such as "customer", "order", "order line" and "shipment". The relationships between entities could be, for instance: each order has a corresponding customer; each shipment has at least one corresponding order line, etc.

In this section, we explain how to model the data perspective of a business process. More precisely, we introduce data constraints to describe the constraints on the data perspective first. Using classes to represent entities and class relationships to represent the relations between entities, we then define a *class model* to describe the data perspective. By defining *valid object models* of class models, i.e., possible instances of class models, we illustrate the semantics of class models.

5.2.1 Data Constraints

Data constraints are employed to describe the relationships between entities on the data perspective. In data modeling languages, cardinalities are often used to form data constraints , e.g., ER models and UML Class models may include cardinality constraints. A cardinality is a set of integers (cf. Chapter 4), e.g., "1..*" denotes the set of all positive integers. The constraint indicated by a relationship between two entities can be specified by two cardinalities. Consider for example a relationship between entities "order" and "order line". Each order should have at least one order line (denoted by the cardinality "1..*") and each order line should have precisely one corresponding order (denoted by the cardinality "1").

The cardinalities are not enough to precisely describe the constraints in some situations. For instance in Figure 5.3, each order line (e.g., *ol*) should be delivered to its corresponding customer at some point in time (e.g., *t*). In this situation, for an order line it is not necessary to "always" have a corresponding delivery (e.g., *ol* has no corresponding delivery before *t*), but it should "eventually" have a corresponding delivery (e.g., *ol* has a corresponding delivery after *t*). Apparently, the cardinalities on edge *r1* cannot precisely describe this situation. Therefore, we add two types of symbols ($\Box$ and $\Diamond$) onto cardinalities (cf. the cardinalities on edge *r2*) to strengthen their expressive power.

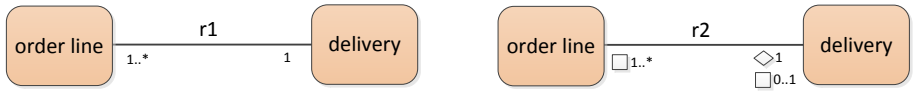


Figure 5.3: Extending cardinality with □ (“always”) and ♢ (“eventually”) symbols.

More precisely, the cardinalities with the square symbol □ are called “always” cardinalities while the cardinalities with the diamond symbol ♢ are called “eventually” cardinalities. “always” cardinalities require cardinalities to be satisfied always. For instance, the “always” cardinality in the constraint that an order line should have “□ 0..1” corresponding delivery, means that at any point in time, the order line should have either zero or one corresponding delivery. In contrast, “eventually” cardinalities require cardinalities to be satisfied eventually. For instance, the “eventually” cardinality in the constraint that an order line should have “♢ 1” corresponding delivery, means that from some point in time onward, the order line should have precisely one corresponding delivery.

Definition 5.1 (Data Constraints) *Let $\mathcal{U}_{Card}$ be the universe of all possible cardinalities. Symbols □ and ♢ are attached before cardinalities to indicate the temporal properties of cardinalities. □ means the cardinality should hold at any point in time and ♢ means the cardinality should hold from some point in time onwards. Examples of data constraints are “□ 0..1” and “♢ 1”.*

notation	allowed correspondence numbers					
	t_s	t_1	...	t_n	...	t_c
□ 1	{1}	{1}	...	{1}	...	{1}
□ 0..1	{0,1}	{0,1}	...	{0,1}	...	{0,1}
□ *	{0,1,2,...}	{0,1,2,...}	...	{0,1,2,...}	...	{0,1,2,...}
♢ 1	{0,1,2,...}	{0,1,2,...}	...	{1}	...	{1}
♢ 0..1	{0,1,2,...}	{0,1,2,...}	...	{0,1}	...	{0,1}
□ 0..1 ♢ 1	{0,1}	{0,1}	...	{1}	...	{1}
□ 1 ♢ 1	{1}	{1}	...	{1}	...	{1}

Table 5.1: Some examples of frequently used extended cardinalities.

It is possible to combine an “always” cardinality and an “eventually” cardinality. A combined cardinality indicates that both the “always” cardinality and

the "eventually" cardinality should be satisfied at the same time. Consider for example the combined cardinality that an order line should have "□ 0..1 ◇ 1" corresponding delivery. It means that the order line should have one corresponding delivery from some point in time onward and either zero or one corresponding delivery before the point in time.

Table 5.1 shows some examples of "always", "eventually" and combined cardinalities. t_s and t_c means the start and complete of a period of time. t_1 corresponds to the first moment after t_s and t_n corresponds to some moment between t_1 and t_c. We use the correspondence number to denote the number of relationship instances a relationship has at some point in time. For instance, if an order has three corresponding invoices at some moment, the correspondence number is three. The allowed correspondence numbers are the numbers of permissible relationship instances a relationship may have.

Note that an "eventually" cardinality only needs to be satisfied from some point in time onward, e.g., t_n (it is possible that t_n is the last moment, i.e., t_c). It does not take effect before t_n, i.e., any (non-negative) integer is allowed before t_n. Consider for example "◇ 1" and "◇ 0..1" in Table 5.1. The allowed correspondence numbers are $\{0,1,2,\ldots\}$ before t_n. A combined cardinality may have the same allowed numbers as an "always" cardinality has. In this case, the combined cardinality can be represented by the "always" cardinality for short, i.e., the "eventually" cardinality can be omitted. For instance, "□ 1 ◇ 1" can be simplified as "□ 1".

5.2.2 Definition of Class Models

In Chapter 3, we defined a data model to describe the structure of the data in database tables, generated by artifact-centric information systems. More precisely, in the context of a database, a data model basically contains classes (representing database tables) and class relationships (representing PK-FK references among tables). The generated data stored in databases correspond to executions of business processes. In other words, the data perspective of a business process is related to the database schema. In order to make it consistent, we still use the notions *classes* and *class relationships* from the data model, but assign them different meanings to describe the data perspective of a business process. In contrast, in the context of a business process, a class represents an entity and a class relationship represents the constraints between two entities.

In Section 5.2.1, we discussed the notations used to describe the data constraints of business processes. More precisely, notations can be viewed as a subset of such mainstream notations, such as ER models [25], UML class models [53],

and Object-Role Models (ORM) [60]. The only particular feature is that cardinality constraints are tagged as "always" (□) or "eventually" (◇). Based on these notations, we define a *class model* to describe the data perspective of a business process.

Definition 5.2 (Class Model) *A* class model *is a tuple* $ClaM = (C, R, \pi_1, \pi_2, \sharp^{\Box}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\Box}_{tar}, \sharp^{\Diamond}_{tar})$, *where*

- $C \in \mathcal{U}_C$ *is a set of classes,*
- $R \in \mathcal{U}_R$ *is a set of relationships* ($C \cap R = \emptyset$),
- $\pi_1 \in R \rightarrow C$ *gives the source class of a relationship,*
- $\pi_2 \in R \rightarrow C$ *gives the target class of a relationship,*
- $\sharp^{\Box}_{src} \in R \rightarrow \mathcal{U}_{Card}$ *gives the source cardinality of a relationship (the constraint should hold at any point in time as indicated by* □*),*
- $\sharp^{\Diamond}_{src} \in R \rightarrow \mathcal{U}_{Card}$ *gives the source cardinality of a relationship (the constraint should hold from some point onwards as indicated by* ◇*),*
- $\sharp^{\Box}_{tar} \in R \rightarrow \mathcal{U}_{Card}$ *gives the target cardinality of a relationship (the constraint should hold at any point in time as indicated by* □*), and*
- $\sharp^{\Diamond}_{tar} \in R \rightarrow \mathcal{U}_{Card}$ *gives the target cardinality of a relationship (the constraint should hold from some point onwards as indicated by* ◇*).*

$\mathcal{U}_{ClaM}$ *is the universe of class models.*

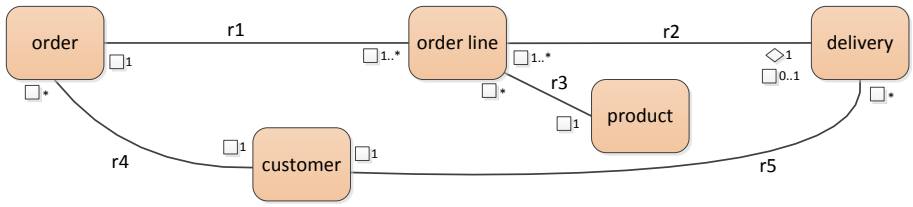


Figure 5.4: An example of a class model.

Figure 5.4 depicts an example of a class model $ClaM = (C, R, \pi_1, \pi_2, \sharp^{\Box}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\Box}_{tar}, \sharp^{\Diamond}_{tar})$. It has five object classes $C = \{order, order\ line, delivery, customer, product\}$ and five relationships $R = \{r1, r2, r3, r4, r5\}$. Relationship *r1* is connecting classes *order* and *order line*: $\pi_1(r1) = order$ and $\pi_2(r1) = order\ line$. For the other relationships, we have: $\pi_1(r2) = delivery$, $\pi_2(r2) = order\ line$, $\pi_1(r3) = product$, and $\pi_2(r3) = order\ line$, etc.

The notation of combined cardinalities (as shown in Table 5.1) is used to specify the cardinalities in Figure 5.4. $\sharp^{\Box}_{src}(r1) = \{1\}$, i.e., for each object in class

order line there is always precisely one corresponding object in *order*. This is indicated by the "□ 1" annotation on the source side (i.e., the *order* side of $r1$) in Figure 5.4. $\sharp^{\Box}_{tar}(r1) = \{1,2,3,\ldots\}$, i.e., for each object in class *order* there is always at least one corresponding object in *order line*. This is indicated by the "□ 1..*" annotation on the target side (i.e., the *order line* side) of $r1$. Not shown are $\sharp^{\Diamond}_{src}(r1) = \{1\}$ ("◇ 1") and $\sharp^{\Diamond}_{tar}(r1) = \{1,2,3,\ldots\}$ ("◇ 1..*") as these are implied by the "always" constraints. On the source side of $r2$ in Figure 5.4, there are two cardinality constraints: $\sharp^{\Box}_{src}(r2) = \{0,1\}$ and $\sharp^{\Diamond}_{src}(r2) = \{1\}$. This means that eventually each order line needs to have a corresponding delivery ("◇ 1"), but the corresponding delivery may not be created before this moment ("□ 0..1"). We only show the "eventually" (◇) cardinality constraints that are more restrictive than the "always" (□) cardinalities in the class model. Obviously, $\sharp^{\Diamond}_{src}(r) \subseteq \sharp^{\Box}_{src}(r)$ and $\sharp^{\Diamond}_{tar}(r) \subseteq \sharp^{\Box}_{tar}(r)$ for any $r \in R$ since constraints that always hold also hold eventually.

Classes can also have attributes and therefore, in principle, the class model should list the names and types of these attributes. We abstract class attributes from this thesis, as well as the notions of hierarchies and subtyping, but they could be added in a straightforward manner by referring to existing techniques. For instance, the data model in object-relational mapping systems can adopt hierarchies and many-to-many relationships, which are then properly flattened in the underlying database. The data model in ontology-based data access (OBDA) systems is at a much higher level of abstraction than the underlying database, and the two layers come with a mapping specification indicating how queries on the database link to classes/relationships in the data model. Note that both such systems adopt data models that are richer than the class model introduced above. These models can be referred to for extending our class models in future .

5.2.3 Semantics of Class Models

A class model describes the data perspective of a business process, by specifying the possible classes and constraints between classes. In this section, we explain the semantics of a class model in terms of its possible instances.

A data model (defined in Chapter 3) describes the structure of data generated by executing a business process. It specifies the constraints from the instance angle. In contrast, the class model describes the business process angle and it specifies the constraints from the same angle. Since class models are related to data models, e.g., they share the notions of classes and class relationships, we also consider *objects* and *object models* as instances of classes and class models,

respectively.

More precisely, in the context of object-centric event data, we use the term "object" to abstract records in database tables and "object model" to represent database states. For example, an object $o = (c, map)$ corresponds to a record in the c table (the values in the record are indicated by map) and an object model $OM = (Obj, Rel, class, objectAttr)$ corresponds to a state of the database at some point in time. In the context of a business process, the object o is considered as an instance of the class c and the object model OM is considered as an instance of a class model.

A class model defines a "space" of possible *object models*, i.e., concrete collections of objects and relations instantiating the class model. A possible object model of a class model should satisfy some rules indicated by the constraints of the class model. For instance, objects in the object model should be classified in classes in the class model. Besides, the cardinalities specified in the class model should be respected by the object model. An object model is *valid* for a class model if it complies with the "always" ($\Box$) cardinalities in the class model. A valid model is also *fulfilled* if the stronger "eventually" ($\Diamond$) cardinality constraints are satisfied.

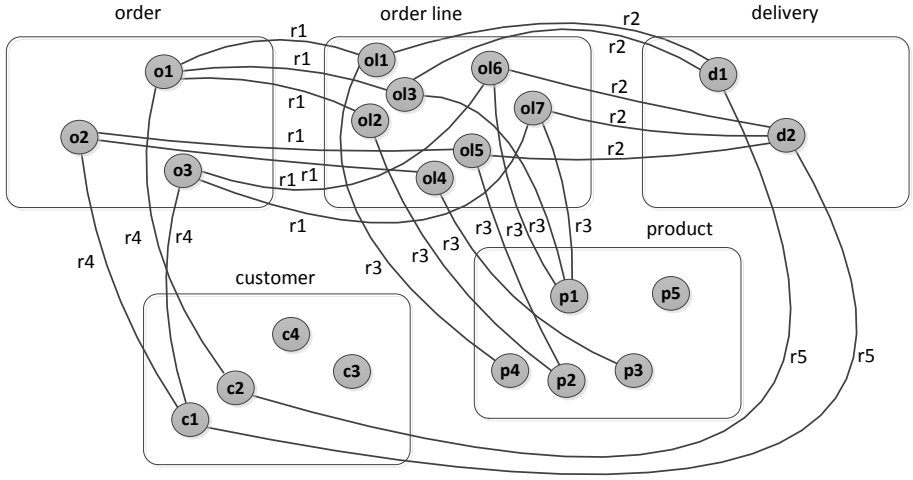


Figure 5.5: An example of an object model corresponding to the class model in Figure 5.4.

Definition 5.3 (Valid and Fulfilled) *Let* $ClaM = (C, R, \pi_1, \pi_2, \sharp^{\square}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\square}_{tar}, \sharp^{\Diamond}_{tar})$ *be a class model and* $OM = (Obj, Rel, class, objectAttr) \in \mathcal{U}_{OM}$ *be an object model. OM is valid for ClaM if and only if*

- *for any* $o \in Obj$*:* $class(o) \in C$*,*
- *for any* $(r, o_1, o_2) \in Rel$*:* $r \in R$*,* $class(o_1) = \pi_1(r)$ *and* $class(o_2) = \pi_2(r)$*,*
- *for any* $r \in R$ *and* $o_2 \in \partial_{\pi_2(r)}(Obj)$*, we have that* [1]

$$|\{o_1 \in Obj \mid (r, o_1, o_2) \in Rel\}| \in \sharp^{\square}_{src}(r), \text{ and}$$

- *for any* $r \in R$ *and* $o_1 \in \partial_{\pi_1(r)}(Obj)$*, we have that*

$$|\{o_2 \in Obj \mid (r, o_1, o_2) \in Rel\}| \in \sharp^{\square}_{tar}(r)$$

A valid object model is also fulfilled if the stronger cardinality constraints hold (these are supposed to hold eventually):

- *for any* $r \in R$ *and* $o_2 \in \partial_{\pi_2(r)}(Obj)$*, we have that*

$$|\{o_1 \in Obj \mid (r, o_1, o_2) \in Rel\}| \in \sharp^{\Diamond}_{src}(r), \text{ and}$$

- *for any* $r \in R$ *and* $o_1 \in \partial_{\pi_1(r)}(Obj)$*, we have that*

$$|\{o_2 \in Obj \mid (r, o_1, o_2) \in Rel\}| \in \sharp^{\Diamond}_{tar}(r)$$

The object model in Figure 5.5 is valid for the class model in Figure 5.4. If we would remove relation $(r1, o1, ol1)$, the model would no longer be valid because an order line should always have a corresponding order. Adding a relation $(r1, o2, ol1)$ would also destroy validity. Both changes would violate the "$\square$ 1" constraint on the source side of *r1*. The object model in Figure 5.5 is not fulfilled because the "$\Diamond$ 1" constraint on the source side of *r2* does not hold. Order lines *ol2* and *ol4* do not (yet) have a corresponding delivery. Adding deliveries for these order lines and adding the corresponding relations would make the model fulfilled.

Definition 5.3 only formalizes simple cardinality constraints involving a binary relation and abstracting from attribute values, i.e., the object model *OM* is simply checked against a class model *ClaM* in terms of "always" and "eventually" cardinalities. In principle, more sophisticated constraints could be considered, e.g., constraints involving attributes or more than two classes. The Object Constraint Language (OCL) [54] could also be used to define more refined constraints.

[1] $\partial_c(Obj) = \{o \in Obj \mid class(o) = c\}$ denotes the whole set of objects in class c.

5.3 Modeling Behavioral Perspective

An OCBC model covers data and behavioral perspectives. It can be considered as extending a class model (describing the data perspective) with an activity model (describing the behavioral perspective). The class model is the backbone of the OCBC model, which is defined in Section 5.2. In this section, we define the activity model to describe the behavioral perspective of a business process.

5.3.1 Behavioral Constraints

In artifact-centric information systems, the behavioral perspective is quite loose and unstructured, which enables users to deal with flexible and dynamic business processes. For instance, after the creation of an order, one can edit, cancel or delete the order, or create an invoice/delivery for the order. Compared with WFM/BPM, artifact-centric information systems provide an "open" environment to operate more freely.

More precisely, on the instance level, the behavioral perspective of a business process is merely a collection of *events* without assuming some case or process instance notion. Note that the orders between events should satisfy some rules indicated by the process. On the model level, the behavioral perspective can be viewed as a set of *activities* and a set of *constraints* between activities without a clear and concrete process. For instance, a constraint may be a temporal restriction on activities, e.g., "display order" activity can happen only after "create order" activity happens. A constraint may also be related to a cardinality. For instance, each "create order" event can have at most one corresponding "delete order" event.

There exist different notations to specify the constraints between activities. In a procedural language like Petri nets, places correspond to constraints: removing a place may allow for more behavior and adding a place can only restrict behavior. Compared with procedural languages like Petri nets, declarative languages are "open" languages (if we consider procedural languages as closed languages), i.e., they allow anything which is not explicitly forbidden by constraints. Therefore, they are more flexible and suitable for modeling business processes in artifact-centric information systems.

In this section, we employ the declarative manner to model the behavioral constraints and refer to the graphical notation inspired by *Declare* [143] (cf. Chapter 2) to depict the constraints. More precisely, a declarative constraint has a reference activity, a target activity and a constraint type which specifies the restriction between the reference events (i.e., events of reference activity) and

target events (i.e., events of target activity). For example, given some *reference event e*, the constraint type requires that the cardinality of the set of target events before or after the reference event lies within a particular range. Based on this idea, a constraint type is defined as a set (corresponding to a particular range) of pairs of integers. In each pair, the first integer indicates the allowed number of target events before the reference event and the second integer indicates the allowed number of target events after the reference event.

Definition 5.4 (Constraint Types) $\mathcal{U}_{CT} = \{X \subseteq \mathbb{N} \times \mathbb{N} \mid X \neq \emptyset\}$ *defines the universe of all possible constraint types. An element of* $\mathcal{U}_{CT}$ *specifies a non-empty set of pairs of integers: the first integer defines the number of target events before the reference event and the second integer defines the number of target events after the reference event.*

constraint type	formalization
response	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after \geq 1\}$
unary-response	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after = 1\}$
non-response	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after = 0\}$
precedence	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before \geq 1\}$
unary-precedence	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1\}$
non-precedence	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 0\}$
co-existence	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before + after \geq 1\}$
non-co-existence	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before + after = 0\}$

Table 5.2: Examples of constraint types (i.e., elements of $\mathcal{U}_{CT}$). The notion is inspired by *Declare*, but formalized in terms of cardinality constraints rather than LTL.

Table 5.2 shows eight examples of constraint types. The corresponding graphical representations of the eight example constraint types are shown in Figure 5.6. Definition 5.4 gives a general notion of *constraint types*. In other words, besides the examples in Table 5.2, one can define any constraint type that can be specified in terms of the *cardinality of preceding and succeeding target events relative to a collection of reference events*.

Note that since declarative constraints are used to describe the loose constraints or rules in flexible business processes, the "before" and "after" in Definition 5.4 do not strictly require target events directly preceding or following the reference event. In other words, a declarative constraint has an influence scope (e.g., a set of events), in which the number of target events before or

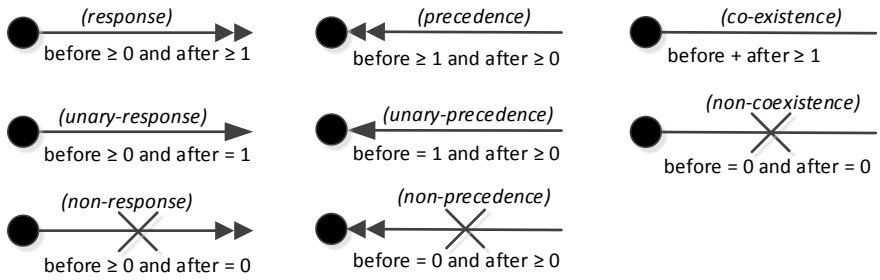


Figure 5.6: Graphical notation for the example constraint types defined in Table 5.2.

after the reference event should satisfy the constraint. In traditional process mining notations, the scope corresponds to a case. Since we do not assume a case notion in our approach, the scope for declarative constraints in OCBC models is identified based on the data perspective (cf. Section 5.4.3).

Definition 5.5 (Behavioral Constraints) *Let $\mathcal{U}_{Con}$ be the universe of possible constraints. Each constraint con $\in \mathcal{U}_{Con}$ has a reference activity, a target activity, a constraint type and an influence scope.*

Definition 5.5 introduces the behavioral constraints in a descriptive way. The influence scope of a constraint is related to the data perspective, which will be illustrated later. Therefore, in this section, we do not consider the influence scope.

Figure 5.7: Two behavioral cardinality constraints: *con1* and *con2*. Inspired by *Declare*, the dot indicates the *reference activity*. The *target activity* is on the other side that has no dot. The constraint type is represented by the arrows.

For instance, Figure 5.7 depicts two behavioral constraint examples *con1* and *con2*. A constraint is represented by an edge with an arrow and a black dot between two rectangles denoting activities. The black dot side corresponds to the reference activity while the other side corresponds to the target activity. The shape of the edge indicates the constraint type. Consider for example the

constraint *con1*. Its reference activity is *a2* (indicated by the black dot), its target activity is *a1* and its constraint type is *ct1* (indicated by the shape of the edge). For *con2*, its reference activity is *a3*, its target activity is *a4* and its constraint type is *ct2*.

5.3.2 Definitions of Activity Models

An *activity model* describes the behavioral perspective of a business process. It can be viewed as a set of *activities* and a set of *constraints* between activities. Since the business processes in artifact-centric information systems are quite flexible, we defined the declarative behavioral constraints in Section 5.3.1 to depict the constraints in these processes. In this section, we define activity models based on declarative constraints.

Definition 5.6 (Activity Model) *An* activity model *is a tuple* $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$*, where*

- $A \subseteq \mathcal{U}_A$ *is a set of activities (denoted by rectangles),*
- $Con \subseteq \mathcal{U}_{Con}$ *is a set of constraints* ($A \cap Con = \emptyset$*, denoted by various types of edges),*
- $\pi_{ref} \in Con \rightarrow A$ *defines the reference activity of a constraint (denoted by a black dot connecting constraint and activity),*
- $\pi_{tar} \in Con \rightarrow A$ *defines the target activity of a constraint (other side of edge), and*
- $type \in Con \rightarrow \mathcal{U}_{CT}$ *specifies the type of each constraint (denoted by the type of edge).*

$\mathcal{U}_{ActM}$ *is the universe of activity models.*

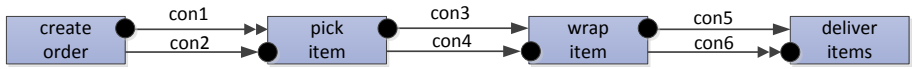


Figure 5.8: An example of an activity model.

An activity model $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ contains a set of activities, i.e., A, and a set of constraints, i.e., Con. Each constraint corresponds to a reference activity, a target activity and a constraint type, indicated by functions π_{ref}, π_{tar}, and $type$, respectively. For instance, Figure 5.8 shows an activity model, which consists of four activities, i.e., $A = \{create\ order, pick\ item, wrap\ item, deliver\ items\}$,

and six constraints, i.e., $C = \{con1, con2, .., con6\}$. Consider *con1* as an example to understand the functions. $\pi_{ref}(con1) = create\ order$, which means that all "create order" events are the reference events for *con1* (indicated by the black dot on the *create order* side of the constraint). $\pi_{tar}(con1) = pick\ item$, which means that all "pick item" events are the target events for *con1*. $type(con1) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after \geq 1\}$ (i.e., the *response* type indicated by the double-headed arrow leaving the black dot). The constraint *con1* means that there has to be at least one "pick item" event after each "create order" event.

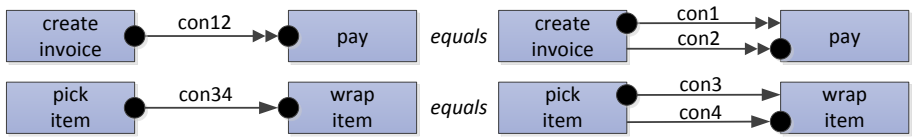


Figure 5.9: An arrow with two black dots (•) can be used as a shorthand. Constraint *con12* (*con34*) corresponds to the conjunction of constraints *con1* and *con2* (resp. *con3* and *con4*).

In an activity model, one arrow may combine two constraints as a shorthand if these two constraints have the same edge type and inverse black dots (i.e., inverse reference activity and target activity). The combined constraint covers the semantics of all its corresponding separate constraints. For example, constraint *con12* in Figure 5.9 indicates that after creating an invoice there is at least one payment, and each payment can cover one or multiple created invoices. The combined constraint *con12* defines the same behavior of the individual constraints *con1* and *con2*. Similarly, constraint *con34* states that after picking an item there is precisely one event to wrap the item, and before wrapping an item there is precisely one event of picking item. The combined constraint *con34* defines the same behavior of the individual constraints *con3* and *con4*. Besides the constraint types present in Figure 5.9, the constraint combination can apply to all constraint types.

Activities can also have attributes and therefore, in principle, the activity model should list the names and types of these attributes. We abstract activity attributes from this thesis, as well as the notions of hierarchies, but they could be added in a straightforward manner.

5.3.3 Semantics of Activity Models

An activity model describes the behavioral perspective of a business process, by specifying the possible activities and constraints between activities. In this section, we explain the semantics of an activity model in terms of the possible instances of the activity model.

An activity model defines a "space" of possible relations (response/precedence and cardinality) between all events in a given set. More precisely, each behavioral constraint in the activity model specifies the allowed relations between each reference event and its corresponding target events. In order to introduce the semantics of behavioral constraints, we first define some notations over a collection of ordered events as a shorthand to refer to events.

Definition 5.7 (Event Notations) *Let $E \subseteq \mathcal{U}_E$ be a set of events ordered by $\preceq$ and related to activities through function act. For any event $e \in E$:*

- $\trianglelefteq_e(E) = \{e' \in E \mid e' \preceq e\}$ *are the events* before *and* including *e.*
- $\trianglerighteq_e(E) = \{e' \in E \mid e \preceq e'\}$ *are the events* after *and* including *e.*
- $\triangleleft_e(E) = \{e' \in E \mid e' \prec e\}$ *are the events* before *e.*[2]
- $\triangleright_e(E) = \{e' \in E \mid e \prec e'\}$ *are the events* after *e.*
- $\partial_a(E) = \{e' \in E \mid act(e') = a\}$ *are the events corresponding to activity $a \in \mathcal{U}_A$.*

In the remainder, these notations can be used in a given set of events, without explicitly stating that the events are ordered and have corresponding activities.

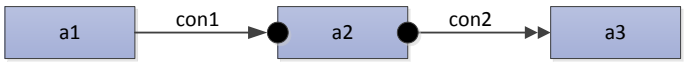


Figure 5.10: An example of an activity model with two constraints.

Figure 5.10 depicts an activity model with two constraints. Given a set of events $E \subseteq \mathcal{U}_E$, the *reference events* for *con1* are all *a2* events, i.e., $\partial_{a2}(E)$. This is indicated by the black dot connecting the *con1* arrow to activity *a2*. Now consider a reference event $e_{ref} \in \partial_{a2}(E)$. The single-headed arrow towards the black dot indicates that e_{ref} should be preceded by precisely one *a1* event. The *a1* events are called *target events* (counterpart of e_{ref} when evaluating the constraint). Formally, constraint *con1* demands that $|\triangleleft_{e_{ref}}(\partial_{a1}(E))| = 1$, i.e., there has to be precisely one *a1* event before e_{ref}.

[2] $e' \prec e$ if and only if $e' \preceq e$ and $e' \neq e$.

The reference events for *con2* are also all *a2* events, i.e., $\partial_{a2}(E)$. Again, this is visualized by the black dot on the *a2* side of the constraint. The double-headed arrow leaving the black dot specifies that any $e_{ref} \in \partial_{a2}(E)$ should be followed by at least one *a3* event. The target events in the context of *con2* are all *a3* events. Formally, $|\rhd_{e_{ref}}(\partial_{a3}(E))| \geq 1$, i.e., there has to be at least one *a3* event after e_{ref}.

Definition 5.8 (Constraint Satisfaction) *Let* $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ *be an activity model, and* $E \subseteq \mathcal{U}_E$ *be a set of events.*

- *E satisfies constraint* $con \in Con$*, denoted as* $E \models con$*, if and only if* $\forall e_{ref} \in \partial_{\pi_{ref}(con)}(E)$:

$$(|\lhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|, |\rhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|) \in type(con).$$

- *E satisfies ActM if and only if E satisfies each constraint* $con \in Con$*.*

Based on the description of the two constraints in Figure 5.10, Definition 5.8 formalizes the requirements which an event set has to meet to satisfy a constraint or an activity model. For a constraint *con*, its reference activity defines the corresponding set of reference events $\partial_{\pi_{ref}(con)}(E)$. For each reference event e_{ref}, it is checked whether the cardinality constraint is satisfied. $\lhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))$ are all target events before the reference event e_{ref} (named precedence target events) and $\rhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))$ are all target events after the reference event e_{ref} (named response target events).

Consider for example the constraint *con1* in Figure 5.10 and the event set E in Figure 5.11 (E contains only correlated events, cf. Section 5.4.3 for how to correlate events). All *a2* events are reference events (i.e., $\{e2, e4\}$) and all *a1* events are target events (i.e., $\{e1, e5\}$). For the reference event *e2*, its precedence target events are $\lhd_{e2}(\partial_{\pi_{tar}(con1)}(E)) = \{e1\}$ and response target events are $\rhd_{e2}(\partial_{\pi_{tar}(con1)}(E)) = \{e5\}$. For the reference event *e4*, its precedence target events and response target events are the same as *e2*. Therefore, the event set E satisfies the constraint *con1*, since for each reference event in E, $(|\lhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|, |\rhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|) = (1,1) \in \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1\}$ (*con1* corresponding to the "unary-precedence" type, cf. Table 5.2).

For another constraint *con2* in Figure 5.10, all *a2* events are reference events (i.e., $\{e2, e4\}$) and all *a3* events are target events (i.e., $\{e3, e6, e7\}$). For the reference event *e2*, its precedence target events are $\lhd_{e2}(\partial_{\pi_{tar}(con2)}(E)) = \emptyset$ and response target events are $\rhd_{e2}(\partial_{\pi_{tar}(con2)}(E)) = \{e3, e6, e7\}$, i.e., $(|\lhd_{e_2}(\partial_{\pi_{tar}(con)}(E))|, |\rhd_{e_2}(\partial_{\pi_{tar}(con)}(E))|) = (0,3)$. For the reference event *e4*, its precedence target events are $\lhd_{e4}(\partial_{\pi_{tar}(con2)}(E)) = \{e3\}$ and response target events are $\rhd_{e4}(\partial_{\pi_{tar}(con2)}(E)) = \{e6, e7\}$, i.e., $(|\lhd_{e_4}(\partial_{\pi_{tar}(con)}(E))|, |\rhd_{e_4}(\partial_{\pi_{tar}(con)}(E))|) = (1,2)$. Therefore, the event

set E satisfies the constraint *con2*, since for each reference event in E, $(|\lhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|, |\rhd_{e_{ref}}(\partial_{\pi_{tar}(con)}(E))|) \in \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after \geq 1\}$ (*con2* corresponding to the "response" type, cf. Table 5.2).

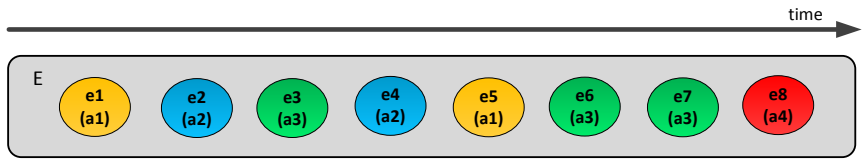


Figure 5.11: An example set of events which are ordered and have corresponding activities.

To sum up, the event set E in Figure 5.11 satisfies the activity model in Figure 5.10, since E satisfies all the constraints (i.e., *con1* and *con2*) in the activity model. Note that the idea of Declare models is that they allow anything which is not explicitly forbidden by constraints. In this sense, a Declare model allows events of any activities which are not in the model, since there are no constraints related to them in the model. In practice, users may want to limit the activities of events. Therefore, we add optional restriction to limit events to only have activities contained by the activity model.

Definition 5.9 (Activity Satisfaction (Optional)) *Let* $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ *be an activity model, and* $E \subseteq \mathcal{U}_E$ *be a set of events (ordered by* $\preceq$ *and related to activities through function act).* E satisfies *activities* A *if and only if* $\forall e \in E : act(e) \in A$.

An event set E satisfies the activities A in an activity model, if the activity of each event e can be found in A, i.e., $act(e) \in A$. For instance, the event set E in Figure 5.11 violates the activity model $ActM$ in Figure 5.10 in terms of activity satisfaction, since the activity of event *e8*, i.e., *a4*, cannot be found in the activities in $ActM$.

In traditional process modeling notations, a constraint is defined for one process instance (case) in isolation. This means that the set E in Definition 5.8 refers to all events corresponding to the same case. As discussed before, the case notion is often too rigid. There may be multiple case notions at the same time, causing one-to-many or many-to-many relationships that cannot be handled using traditional monolithic process models. In OCBC models, the case notion is removed, and the set E is derived by correlating events based on the data perspective (cf. Section 5.4.3).

5.4 OCBC: Integration of Data and Behavioral Perspectives

In Section 5.2, we focused on modeling the data perspective of a business process, by structuring entities and formalizing cardinality constraints on class models (i.e., classical data modeling). In Section 5.3, we modeled the behavioral perspective, i.e., control-flow modeling, and formalized behavioral constraints to depict the restriction between each reference event and its corresponding target events. In this section, we combine *both perspectives* to form a complete *OCBC model* by adding the interactions between activities and classes.

5.4.1 Relating Activities to Classes

The data constraints (i.e., class relationships) connect classes in the class model while the behavioral constraints correlate activities in the activity model. These two types of constraints serve as a bridge to associate elements for each individual perspective. Similarly, in order to integrate the data perspective and behavioral perspective, we define a new type of constraints between activities and classes.

Consider a simple scenario (which is a fragment of a business process related to payments for tickets booked online) to understand the constraints between activities and classes. The scenario can be summarized as the following rules:

- Each ticket always refers to at most one payment. It is possible that the ticket has no payment when it is booked.
- Each ticket eventually refers to precisely one payment. A ticket is valid only when it is paid and the payment cannot be split into multiple ones.
- Each payment refers to one or more tickets. A payment can be made for one ticket after the ticket is booked. It is possible that one payment covers multiple tickets.

Figure 5.12(a) depicts a model to describe this scenario. The model contains an activity "pay" and a class "ticket" and can be considered as a fragment of a larger OCBC model which describes the whole business process. The constraint (or relationship) between "pay" and "ticket" is specified by three cardinalities, an "always" cardinality (corresponding to the first rule) and an "eventually" cardinality (corresponding to the second rule) on the activity side, and a cardinality (without $\Box$ or $\Diamond$ symbol) on the class side (corresponding to the third rule). A fragment of the XOC event log generated in the scenario, i.e., some payment events (of activity "pay") and corresponding object models (only with "ticket" objects), is illustrated in Figure 5.12(b). Payment event *p1* refers to

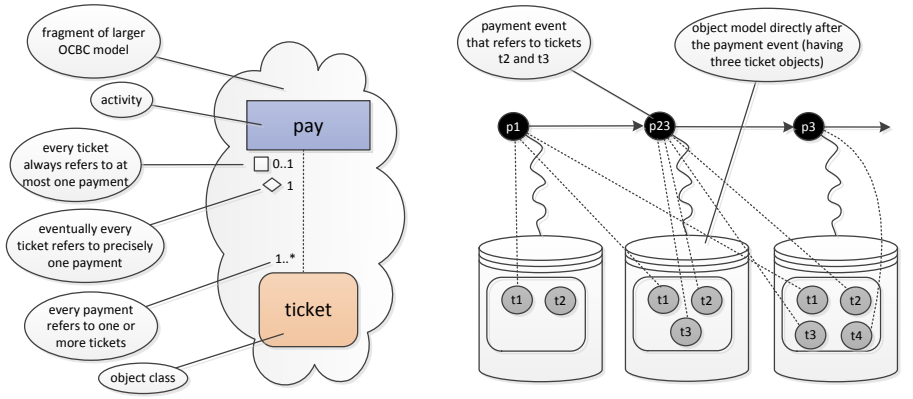


Figure 5.12: Illustrating cardinality constraints between activities and classes.

ticket *t1*, event *p23* refers to tickets *t2* and *t3*, and event *p3* refers to ticket *t4*.

These three cardinalities shown in Figure 5.12(a), i.e., "□ 0..1" (every ticket always refers to at most one payment), "♢ 1" (eventually every ticket refers to precisely one payment), and "1..*" (every payment refers to one or more tickets) are consistent with the observed events and object models. For instance, the object *t2* has no corresponding payment event at the moment when *p1* happens ("□ 0..1") and from the moment when *p23* happens onwards, it has precisely one corresponding payment event ("♢ 1"). Payment events $p1$ and $p23$ refer to one ticket (*t1*) and two tickets (*t2* and *t3*), respectively, when they happen ("1..*").

Indicated by the discussion of the example scenario, we also use the cardinality constraints to express the restriction between activities and classes (similar to cardinality constraints in a class model). More precisely, the constraints on the activity side define how many events there should be for each object. Since the object model is evolving, there are two types of constraints: constraints that should hold *at any point in time* from the moment the object exists (□) and constraints that should *eventually* hold (♢). In contrast, the constraint on the class side defines how many objects there need to be for each event when the event occurs. Since such a constraint is only for a specific moment, there is no need to add "always" or "eventually" symbol on the constraint.

Definition 5.10 (AOC Relationships) *Let $A \in \mathcal{U}_A$ be a set of activities, $C \in \mathcal{U}_C$ be a set of classes. $AOC \subseteq A \times C$ is a set of AOC relationships between* Activities

and (Object) Classes. *For convenience, we define three functions to refer to the cardinalities on the relationships.*

- $\sharp_A^{\Box} \in AOC \rightarrow \mathcal{U}_{Card}$ *gives the source "always" cardinality of an AOC relationship (activity side, the constraint should hold at any point in time as indicated by* $\Box$*),*
- $\sharp_A^{\Diamond} \in AOC \rightarrow \mathcal{U}_{Card}$ *gives the source "eventually" cardinality of an AOC relationship (activity side, the constraint should hold from some point onwards as indicated by* $\Diamond$*), and*
- $\sharp_{OC} \in AOC \rightarrow \mathcal{U}_{Card}$ *gives the target cardinality of an AOC relationship (object class side).*

$\mathcal{U}_{AOC}$ *is the universe of AOC relationships.*

The constraints between activities and classes illustrated above are defined as AOC relationships in Definition 5.10 to distinguish with other types of constraints or relationships. Besides, we also define three functions $\sharp_A^{\Box}$, $\sharp_A^{\Diamond}$ and $\sharp_{OC}$ to refer to the three cardinalities on the relationships. For instance, there is one AOC relation in Figure 5.12(a), i.e., $AOC = \{(pay, ticket)\}$, and $\sharp_A^{\Box}((pay, ticket)) = \Box$ 0..1, $\sharp_A^{\Diamond}((pay, ticket)) = \Diamond$ 1 and $\sharp_{OC}((pay, ticket)) = 1..*$.

Note that when we say the moment when an event occurs, it refers to the moment just after the event occurs. For instance, if an object is added by an event, we say the object exists when the event occurs (which means in reality that the object exists just after the event occurs). We do not distinguish the moment just before the event and the moment just after the event. As a result, no matter if a ticket is created by an event just before a payment event or it is created by the payment event, we consider that the ticket exists when the payment event occurs.

5.4.2 Definition of OCBC Models

The AOC relationships between activities and classes introduced in Section 5.4.1 provide the integration needed to connect activity models to class models. Next, we define *Object-Centric Behavioral Constraint* (OCBC) models, which relate behavior and data structure, through a combination of control-flow modeling and data/object modeling.

Definition 5.11 (Object-Centric Behavioral Constraint Model) *An* object-centric behavioral constraint model *is a tuple* $OCBCM = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$*, where*

- $ClaM = (C, R, \pi_1, \pi_2, \sharp_{src}^{\Box}, \sharp_{src}^{\Diamond}, \sharp_{tar}^{\Box}, \sharp_{tar}^{\Diamond})$ *is a class model (Definition 5.2),*

- $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ *is an activity model (Definition 5.6),*
- C, R, A *and* Con *are pairwise disjoint (no name clashes),*
- $AOC \subseteq A \times C$ *is a set of AOC relationships, and* $\sharp_A^{\square}$, $\sharp_A^{\diamond}$ *and* $\sharp_{OC}$ *specify the three cardinalities on the AOC relationships (Definition 5.10),*
- $crel \in Con \rightarrow C \cup R$ *indicates the event correlation pattern (to identify the scope) for each behavioral constraint, satisfying the following conditions for each* $con \in Con$*:*
  - $\{(\pi_{ref}(con), c), (\pi_{tar}(con), c)\} \subseteq AOC$ *if* $crel(con) = c \in C$*, and*
  - $\{(\pi_{ref}(con), \pi_1(r)), (\pi_{tar}(con), \pi_2(r))\} \subseteq AOC$ *or* $\{(\pi_{ref}(con), \pi_2(r)), (\pi_{tar}(con), \pi_1(r))\} \subseteq AOC$ *if* $crel(con) = r \in R$.

$\mathcal{U}_{OCBCM}$ *is the universe of OCBC models.*

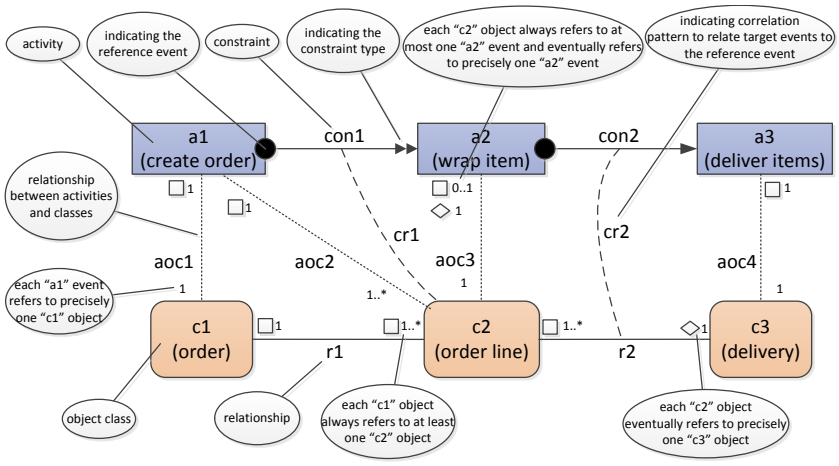


Figure 5.13: An example model illustrating the main ingredients of an OCBC model.

An *Object-Centric Behavioral Constraint* (OCBC) model $OCBCM = (ClaM, ActM, AOC, \sharp_A^{\square}, \sharp_A^{\diamond}, \sharp_{OC}, crel)$ includes a class model $ClaM$ (to describe objects/-data) and an activity model $ActM$ (to describe behavior). These two models are related through AOC relationships AOC and functions $\sharp_A^{\square}$, $\sharp_A^{\diamond}$, $\sharp_{OC}$, and $crel$. We use the OCBC model example in Figure 5.13 to clarify these concepts. In order to make it easier to understand, we add a scenario on the OCBC model, e.g., *a2* corresponds to the activity "wrap item" and *c2* corresponds to the class "order line".

In Figure 5.13, the class model *ClaM* includes three classes (i.e., $C = \{order, order\ line, delivery\}$) and two class relationships (i.e., $R = \{r1, r2\}$). *r1* specifies a one-to-many relationship between orders and order lines, and *r2* indicates that each order line eventually has a corresponding delivery and a delivery always contains one or multiple order lines. The activity model *ActM* includes three activities (i.e., $A = \{create\ order, wrap\ item, deliver\ items\}$) and two behavioral constraints (i.e., $Con = \{con1, con2\}$). The behavioral constraint *con1* means that each "create order" event should be followed by at least one "wrap item" event, and *con2* indicates that each "wrap item" event should be followed by precisely one "deliver items" event.

AOC relationships relate activities and classes on the model level, which indicate the possible references between events and objects on the instance level. For instance, $AOC = \{(a1, c1), (a1, c2), (a2, c2), (a3, c3)\}$ in Figure 5.13 indicates that *a1* events may potentially refer to *c1* and *c2* objects, but not to *c3* objects because $(a1, c3) \notin AOC$. Note that an activity can refer to multiple classes and a class can be referred to by multiple activities. For instance, *a1* refers to both *c1* and *c2*, and *c2* is referred to by both *a1* and *a2*. This many-to-many relations between activities and classes in OCBC models (on the model level) correspond to the many-to-many relations between events and objects in XOC event logs (on the instance level).

Each AOC relationship has an "always" cardinality and an "eventually" cardinality on the activity side, specified by functions $\sharp_A^{\Box}$ and $\sharp_A^{\Diamond}$, respectively. It also has a cardinality on the class side, specified by function $\sharp_{OC}$. For instance, $\sharp_A^{\Box}(a2, c2) = \Box\ 0..1$ (corresponding to the $\Box$ annotation on the *a2* side of the line connecting activity *a2* and class *c2*), which means that each order line should be wrapped at most once. Note that an "order line" object does not need to have a corresponding "wrap item" event when it is created. $\sharp_A^{\Diamond}(a2, c2) = \Diamond\ 1$ (corresponding to the $\Diamond$ annotation on the *a2* side of the same line), which means that eventually each order line should be wrapped once. $\sharp_{OC}(a2, c2) = 1$ (corresponding to the annotation on the *c2* side of the same line), which means that each "wrap item" event precisely wraps one order line.

Function *crel* assigns a class *c* or a class relationship *r* to a behavioral constraint *con*, which indicates the event correlation pattern for the constraint, as shown in Figure 5.14. In other words, *crel* defines the *scope* of each behavioral constraint thereby relating reference events to corresponding target events. $crel(con)$ specifies how events need to be correlated (i.e., through a class *c* or a class relationship *r*) when evaluating constraint *con*. This is needed because we do not assume a fixed case notion and different entities may interact. For instance, in Figure 5.13 $crel(con1) = c2$ (indicated by the line *cr1* connecting *con1*

and *c2*) and *crel*(*con2*) = *r2* (indicated by the line *cr2* connecting *con2* and *r2*). As illustrated by Figure 5.14 we basically consider two types of event correlation patterns: (a) $crel(con) = c \in C$, i.e., the target events are related to the reference event through shared objects of class *c*, (b) $crel(con) = r \in R$, i.e., the target events are related to the reference event through relations of type *r* (in any direction). In both cases, we navigate through the object model to find target events for a given reference event. In the next section, we will illustrate more details about how to correlate events in the context of OCBC models.

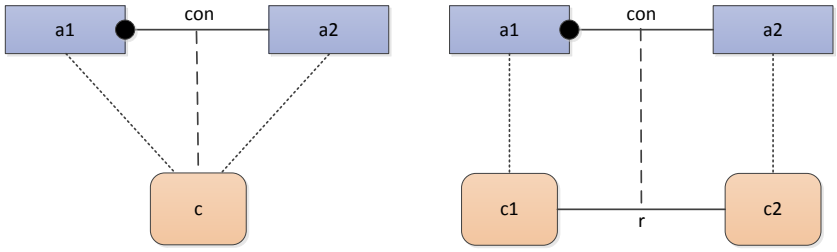


Figure 5.14: Two types of event correlation patterns for constraints: (a) $crel(con) = c \in C$ and (b) $crel(con) = r \in R$.

We have now introduced all the modeling elements used in Figure 5.13. In OCBC models, annotations of the type "♢ *" and "□ *" are omitted from the diagram because these impose no constraints. Also implied constraints can be left out, e.g., "□ 1..*" implies "♢ 1..*".

5.4.3 Semantics of OCBC Models

An OCBC model contains a data model, an activity model and AOC relationships in between. The semantics of data models, activity models and AOC relationships are illustrated in Section 5.2.3, Section 5.3.3 and Section 5.4.1, respectively. In this section, we illustrate the semantics of the OCBC model as a whole. More precisely, since different elements (perspectives) in an OCBC model influence each other, we focus on explaining the semantics in terms of the interactions (i.e., the semantics added after combining the two perspectives) [126].

Note that in Section 5.3.3 the semantics of behavioral constraints in an activity model is explained in terms of a given event set E. In traditional process

modeling notations, the event set E refers to all events corresponding to a specific case. However, in OCBC models, we do not assume the case notion, and the event set E is identified based on the data perspective. More precisely, the function $crel$ in an OCBC model specifies two different types of patterns to identify the scope for each constraint. In other words, these patterns correlate target events to each reference event, which are called event correlation patterns.

Definition 5.12 (Event Correlation Patterns) *Let $OCBCM = (ClaM, ActM, AOC, \sharp_A^{\square}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model, where $ClaM = (C, R, \pi_1, \pi_2, \sharp_{src}^{\square}, \sharp_{src}^{\Diamond}, \sharp_{tar}^{\square}, \sharp_{tar}^{\Diamond})$ is a class model and $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ is an activity model. A correlation pattern of OCBCM is a tuple $(a_{ref}, a_{tar}, cr) \in A \times A \times C \cup R$ where $a_{ref} \neq a_{tar}$ and*

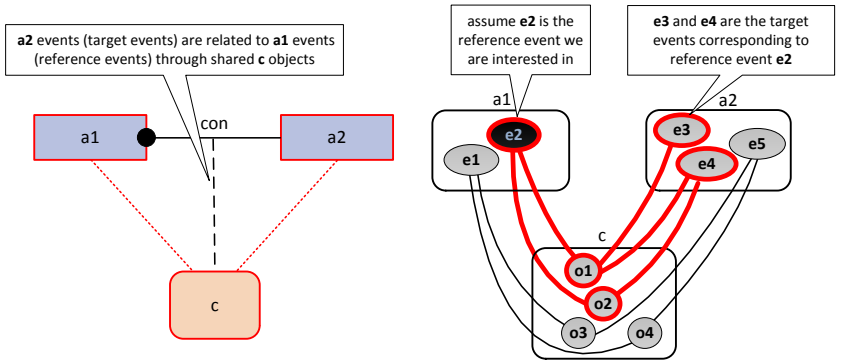
- $\exists c \in C : cr = c \wedge \{(a_{ref}, c), (a_{tar}, c)\} \subseteq AOC$, *or*
- $\exists r \in R : cr = r \wedge (\{(a_{ref}, \pi_1(r)), (a_{tar}, \pi_2(r))\} \subseteq AOC \vee \{(a_{ref}, \pi_2(r)), (a_{tar}, \pi_1(r))\} \subseteq AOC)$.

$\mathcal{U}_P$ is the universe of correlation patterns.

A correlation pattern consists of two activities (a reference activity and a target activity) and a class or class relationship, which serves as a "bridge" to connect these two activities. In an OCBC model, a behavioral constraint corresponds to a correlation pattern which indicates how to identify the target events of each reference event for the constraint. For instance, a constraint *con* corresponds to the pattern $(\pi_{ref}(con), \pi_{tar}(con), crel(con))$. As shown in Figure 5.14, the correlation patterns are divided into two types based on the type of the "bridge", i.e., $crel(con)$. Next, we discuss the details about these two types of patterns and how to correlate events based on them.

If $crel(con) = c \in C$, the constraint *con* corresponds to a V pattern as shown in Figure 5.15(a), since the event correlation pattern consists of two activities linked to a class, which looks like the letter "V". In this situation, the target events are related to the reference events through shared objects of c. More precisely, let e_{ref} be a reference event for constraint *con*. e_{ref} refers to one or more c objects. The target events of e_{ref} for *con* are those $\pi_{tar}(con)$ events referring to (at least) one of these objects. For instance, Figure 5.15(a) shows a constraint *con* where $crel(con) = c$, *a1* is the reference activity, *a2* is the target activity and both *a1* and *a2* refer to the class *c*. Its corresponding correlation pattern is a V pattern $(a1, a2, c)$. Figure 5.15(b) shows the event correlation on the instance level. For the reference event *e2*, which refers to objects *o1* and *o2* (of class *c*), *e3* is one of its target events since *e3* also refers to the object *o1*. Similarly, *e4* is also a target event of *e2*.

If $crel(con) = r \in R$, the constraint *con* corresponds to a U pattern as shown in Figure 5.16(a), since the event correlation pattern consists of two activities



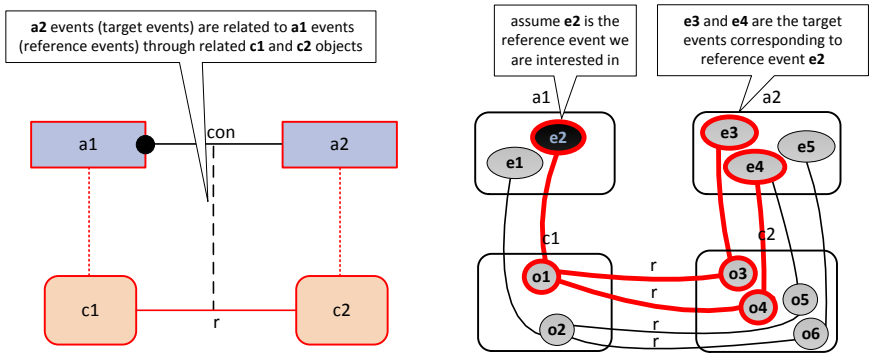
(a) **V pattern**: two activities refer to a shared class

(b) the reference event and target events are related through shared objects

Figure 5.15: Given a reference event for a constraint with $crel(con) = c \in C$ we navigate to the target events through shared object references.

linked to a class relationship, which looks like the letter "U". In this situation, the target events are related to the reference events through objects connected by object relations of relationship r in the object model. More precisely, let e_{ref} be a reference event for constraint *con*, which refers to a set of objects O. O' denotes the objects which are related to (at least) one object in O through object relations of r. The target events of e_{ref} for *con* are those $\pi_{tar}(con)$ events referring to (at least) one object in O'. Figure 5.16(a) shows a constraint *con* where $crel(con) = r \in R$, *a1* is the reference activity and refers to the class *c1*, *a2* is the target activity and refers to the class *c2*, and *c1* is related to *c2* through a class relationship r. Consider the events in Figure 5.16(b) as an example. For the reference event *e2*, which refers to an object *o1* (of class *c1*), *e3* is one of its target events since *e3* refers to an object *o3* (of class *c2*), which is connected to *o1* through an object relation of r. Similarly, *e4* is also a target event of *e2*.

It is essential to understand that the scoping of events considered for a constraint is done through the object model. This provides a tight integration between behavior and structure (i.e., data). Moreover, the approach is much more general and more expressive than classical approaches where events are correlated through cases. Usually, classical process models (both procedural and declarative) describe the lifecycle of a process instance (i.e., case) in isolation. This implies that events are partitioned based on case identifiers and different cases cannot



(a) **U pattern**: two activities refers to two related classes

(b) the reference event and target events are related through related objects

Figure 5.16: Given a reference event for a constraint with $crel(con) = r \in RT$, we navigate to the target events through related objects connected by relations of r in the object model.

share events. Hence, one-to-many and many-to-many relationships cannot be modeled (without putting instances in separate subprocesses, artifacts or proclets). In contrast, by removing the restriction of the case notion, an OCBC model can express *complex interactions between a variety of objects in a single diagram*.

Note that although OCBC models are suitable for modeling processes with multiple instances, traditional single-instance modeling approaches can still be mimicked by using an object model having one object class *case* and $crel(con) =$ *case* for each constraint *con*. Figure 5.17 sketches this situation and illustrates that the classical view on process behavior is very limited, since complex relationships cannot be captured, and the link to data/object models is missing.

The above description illustrates the semantics of OCBC models in terms of event correlation. Next, we explain the semantics of OCBC models in terms of the unified cardinality constraints. In an OCBC model, data constraints in a data model consist of cardinality constraints and “always”/“eventually” ($\Box/\Diamond$) symbols. The attached symbols strengthen the cardinality constraints by indicating that they should be satisfied always or eventually. In contrast, behavioral constraints in an activity model consist of cardinality constraints and “before” and “after” types. The attached types strengthen the cardinality constraints by indicating that they should be satisfied before or after reference

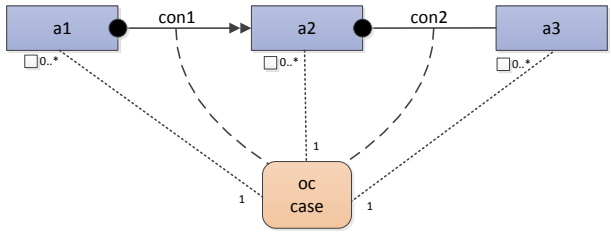


Figure 5.17: An OCBC model mimicking the classical situation where behavior needs to be straightjacketed in isolated process instances (i.e., cases).

events. The constraints of AOC relationships are similar to data constraints, which also consist of cardinality constraints with or without "always"/"eventually" (□/♢) symbols. Since all constraints use cardinality constraints in essence, an OCBC model also allows us to reason about behavior and data in a *unified* manner. For instance, in the OCBC model shown in Figure 5.13, adding a new *oc2* object implies the occurrence of at least one corresponding *a1* event to satisfy the cardinality constraint "♢ 1..*", i.e., an *obligation* is created.

Up to now, we have explained all the semantics of an OCBC model, i.e., the semantics of each part and the semantics of the interactions (e.g., event correlations) when all parts are combined as a whole. Next, we describe a possible execution of the process represented by the OCBC model in Figure 5.13, based on the semantics of the OCBC model. A "create order" event can happen at any time since there are no behavioral constraints adding precondition on the "create order" activity. The "create order" event creates an order (*aoc1*) and multiple order lines (*aoc2*). The order lines correspond to the order through *r1*. Then each order line is wrapped once (*aoc3*), i.e., there are several "wrap item" events following the "create order" event (*con1*). After some time, all the order lines are delivered to the customer. On the data perspective, each "order line" object has a corresponding "delivery" object (*r2*) and on the behavioral perspective, each "wrap item" event has precisely one corresponding "deliver items" event (*con2*).

5.5 Evaluation: Modeling a Recruitment Process

In this section, we evaluate the OCBC modeling language by comparing it with other modeling languages in terms of describing business processes. More precisely, we use the OCBC model and some other related models to describe a given business process, and then compare these models to see which one can best describe the given business process.

Since the recruitment process is quite common and significant for organizations, we choose this process as the given business process to compare models. The recruitment process covers all steps a company goes through, in order to recruit a person for a job position. It often includes job planning, posting positions online, receiving applications, interviews and selections. The process may be very large and complex in some situations. Figure 5.18 shows an easy example which is a part of the whole recruitment process.

Figure 5.18: A recruitment process example.

In order to make the comparison easy to understand, the given process for modeling only involves the significant steps (e.g., applications, interviews and employee selections), which are introduced as follows. According to a specific need, an organization may create a position and recruit a person for the position. People who are interested in such a position can apply for it, but need to register first. Applications for a position are only considered in the period between opening the position and closing the application process for the position. An application may be followed by at most five reference checks and at most two interviews. In the end, one person is selected and subsequently hired for the position.

5.5.1 Modeling Using OCBC Models

First, we use the OCBC modeling language proposed in this chapter to describe the recruitment process. Figure 5.19 shows a designed OCBC model, which

describes the data perspective of the process at the bottom, the behavioral perspective at the top and the interactions in the middle.

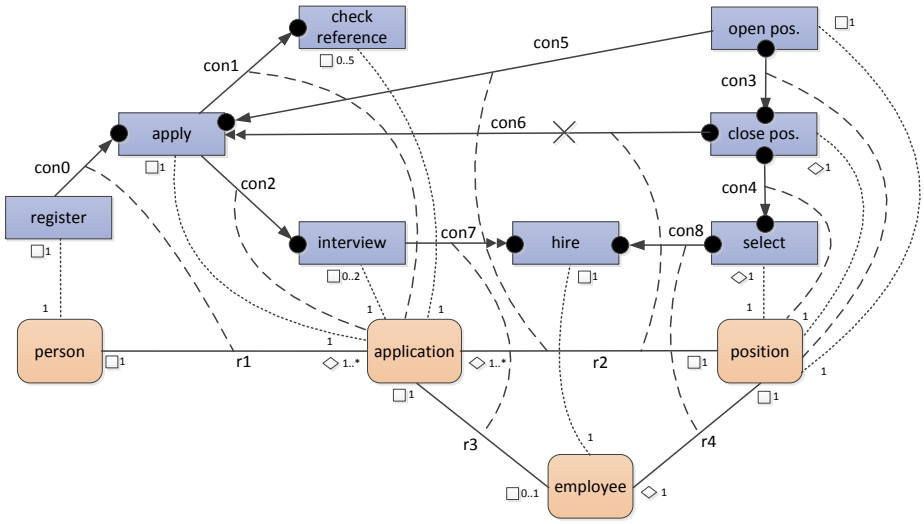


Figure 5.19: An OCBC model modeling the recruitment process.

On the data perspective, there are four classes in the OCBC model: “person”, “application”, “position” and “employee”, which correspond to four entities involved in the process. For instance, the “person” class corresponds to the people who want to apply for the position, the “application” class corresponds to the concrete documents submitted by people for the position, etc. The cardinality constraints in the data model, i.e., *r1*, *r2*, *r3* and *r4*, specify the constraints between the four classes as follows:

- *r1* indicates that each person eventually applies for at least one position (i.e., submits at least one application) and each application always refers to precisely one person;
- *r2* shows that each application always refers to precisely one position and for every position there will eventually be at least one application;
- *r3* indicates that each employee refers to precisely one application and each application refers to at most one employee, and
- *r4* means that each employee refers to precisely one position and each position will eventually refer to one employee.

The interactions present how activities relate to classes. More precisely, there is a one-to-one correspondence between registrations (activity "register") and persons (class "person"). Activities "apply", "check reference", and "interview" each refer to the class "application". An event of activity "apply" creates one new "application" object, which always corresponds to at most two "interview" events and at most five "check reference" events. Activities "open pos.", "close pos.", and "select" each refer to the class "position". An event of activity "open pos." creates one new "position" object, which eventually corresponds to precisely one "close pos." event and "select" event. There is also a one-to-one correspondence between hirings (activity "hire") and employees (class "employee").

The behavioral perspective of the process is defined by eight activities ("register", "apply", "check reference", "interview", "hire", "open pos.", "close pos.", "select") and nine behavioral constraints (*con0,con1,...,con8*). The details of the constraints are as follows:

- Constraint *con0* specifies that every application should be preceded by precisely one corresponding registration (unary-precedence constraint).
- Constraint *con1* specifies that every reference check should be preceded by precisely one corresponding application (unary-precedence constraint).
- Constraint *con2* specifies that every interview should be preceded by precisely one corresponding application (unary-precedence constraint).
- Constraint *con3* combines a unary-response and a unary-precedence constraint stating that opening a position should be followed by the closing of the application process and the closing should be preceded by the opening of the position.
- Constraint *con4* also combines a unary-response and a unary-precedence constraint stating that the two related activities are executed in sequence.
- Constraint *con5* specifies that applications for a position need to be preceded by the opening of that position.
- Constraint *con6* specifies that after closing a position there should not be any new applications for this position (non-response constraint).
- Constraint *con7* specifies that every hire needs to be preceded by at least one interview with the candidate applying for the position (precedence constraint).
- Constraint *con8* combines a unary-response and a unary-precedence constraint stating that the two related activities are executed in sequence.

5.5.2 Modeling Using Declare Models

In this section, we use other models to describe the process. Since OCBC models employ a declarative manner for the behavioral perspective, we first compare them with Declare models.

The corresponding Declare model of an OCBC model can be considered as the behavioral perspective of the OCBC model, i.e., the remaining part after removing the data model and interactions (i.e., AOC relationships) from the OCBC model. Figure 5.20 shows a Declare model which describes the same recruitment process. Due to the lack of a data model, it is obvious that the Declare model can not describe the data perspective, e.g., the structure of the entities involved in the process.

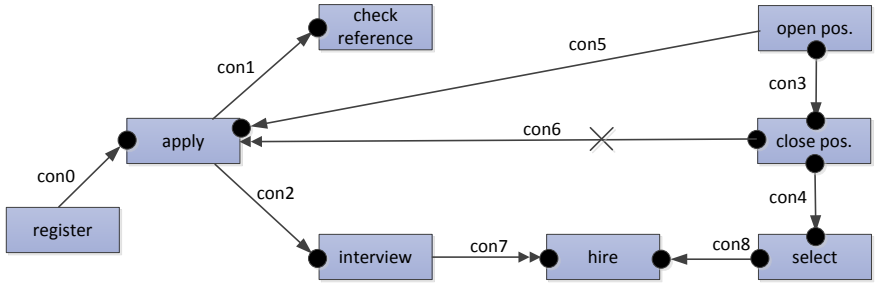


Figure 5.20: A Declare model describing the recruitment process.

On the behavioral perspective, OCBC and Declare models both employ a declarative manner. The most significant difference is that Declare models assume a case notion (i.e., the semantics of constraints in a Declare model is defined in the context of a case notion) while OCBC models do not. When a process contains several artifacts/sub-processes, there often exist multiple case notions (each one corresponding to an artifact/sub-process) and a case notion for the whole process is not clear or inexistent. In this situation, we need to identify a case notion for a specific constraint based on the domain knowledge about the process to interpret the constraint and the case notion changes for different constraints. When the domain knowledge is weak or absent, identifying a proper case notion is difficult and the interpretation of constraints (i.e., the semantics of constraints) could be very confusing and imprecise.

Consider for example the “unary-precedence” constraint *con1* between activities “apply” (target activity) and “check reference” (reference activity) in

Figure 5.20. It indicates that activity "check reference" can be executed only after a corresponding "apply" event. With the domain knowledge, we can know that the constraint works in the context of the case of an "application", i.e., an application can be checked only after an "apply" event which creates the application. In contrast, the constraint *con3* (combination of a "unary-precedence" constraint and a "unary-response" constraint) between activities "open pos." and "close pos." takes effect in the context of the case of a "position", i.e., opening a position should be followed by the closing of the application process and the closing should be preceded by the opening of the position. Apparently, constraint *con1* and constraint *con3* are interpreted choosing different case notions in the process, since these two constraints are in different parts (i.e., sub-processes) of the process.

The examples (i.e., constraints *con1* and *con3*) discussed above show that we have to identify case notions and change case notions accordingly to interpret constraints in different sub-processes, which is difficult in some situations. Besides, we also have problems to interpret the constraints between two different sub-processes. Consider for example the constraint *con6* between activities "close pos." and "apply" from two different sub-processes. Even though we know the case notions for "apply" and "close pos." are "application" and "position", respectively, it is not clear which one we should choose for interpreting this constraint.

Moreover, in real business processes, an activity can be involved in multiple artifacts/sub-processes. For instance, the "apply" activity can (i) make an application to a position and (ii) apply the reimbursement for the cost of attending a interview. In this situation, the "apply" activity is involved in two artifacts/sub-processes, i.e., it corresponds to two case notions. If we have another activity which also has one or more case notions, it is difficult to select a proper case notion and interpret the constraint between these two activities based on the selected case notion.

5.5.3 Modeling Using Workflow Nets

Workflow nets (WF-nets) are commonly used to describe business processes in academia such as the process mining domain. Therefore, we describe the recruitment process with Workflow nets in this section.

Note that there exist one-to-many and many-to-many relationships between different entities in the recruitment process. For instance, a person can submit multiple applications to apply for multiple positions, and a position can be applied by multiple persons. Therefore, there are multiple local case notions

(each of which covers a part of the process) and a global case notion (which covers the whole process) is inexistent. However, a Workflow net requires a global case notion to describe the whole process, which forces one to select a local case notion as the global one. As a result, the process is flattened by the selected case notion. i.e., a Workflow net only shows a specific view of the case notion.

First, we consider the most intuitive situation, where "application" is selected as the case notion. Since one application only corresponds to precisely one person and one position, the view of the process based on the "application" case notion does not have multiple instances of "person" and "position". Figure 5.21 shows the corresponding WF-net. More precisely, after a position is opened ("open pos."), a person can register ("register") and then apply ("apply") for this position. Each application can be checked ("check reference") several times, and meanwhile the corresponding person can be interviewed several times ("interview"). We use a silent/implicit transition t to form a loop to represent the multiple executions of activity "check reference" or "interview". After that, the application process for the position is closed ("close pos.") and then (i) this application is selected ("select") and the related person is hired ("hire") or (ii) this application is not selected and the related person is not hired (indicated by the silent transition). Note that Figure 5.21 also takes into consideration the data perspective, which contains four business objects, i.e., person, application, position and employee. The dotted arrows between activities and business objects indicate interactions between them. For instance, the arrow from "register" to "person" indicates that "register" writes some information into "person".

Next, we select the "position" as the case notion. Since multiple persons can apply for a position (i.e., there is a one-to-many relationship between "position" and "person"), there could be multiple registrations and applications after opening a position. Figure 5.22 shows the WF-net selecting "position" as the case notion. In order to describe the multiple registrations for one position, we add a loop for the application part. With the help of the added loop, Figure 5.22 can present the one-to-many relationship between positions and applications. However, it can not really model the parallel relation between multiple instances of "application". The parallel relation means that different instances can happen at the same time (in an overlapping manner). For instance, after a registration of one instance, another registration of a different instance can happen directly after the first registration. However, Figure 5.22 only allows the occurrence of another application after the first application finishes.

Figure 5.23 presents another Workflow net, which can model the parallel relations between multiple instances of "application". More precisely, after

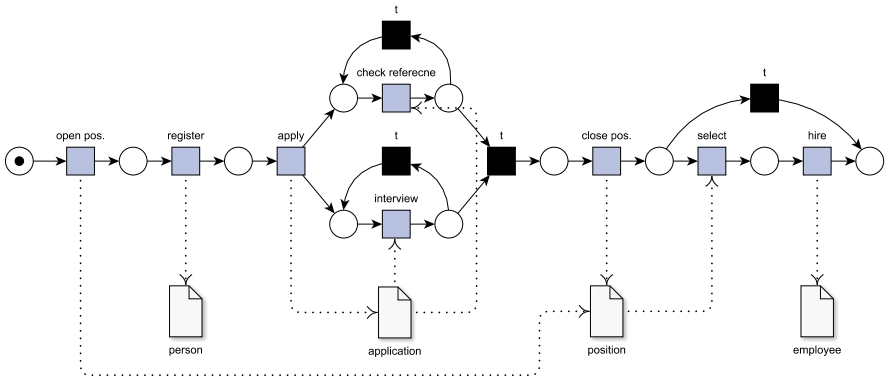


Figure 5.21: A Workflow net modeling the recruitment process selecting “application” as the case notion.

opening the position, a mandatory registration follows. Then any activity of “register”, “apply”, “check reference” and “interview” can happen at any order and for any number of times (before the position is closed). However, this Workflow net fails to describe the structure of the process precisely. For instance, the sequential order between “apply” and “check reference”/“interview” is lost.

In summary, based on a specific case notion, a Workflow net can only describe the recruitment process from an angle corresponding to the selected case notion. The one-to-many and many-to-many relationships in the process are flattened in Workflow nets. Besides, Workflow nets have problems to deal with the multiple instances, either failing to really describe the parallel relations or resulting in too generic structures (i.e., some constraints are lost and too much behavior is allowed).

5.5.4 Modeling Using BPMN Diagrams

BPMN diagrams are widely used in industry. In this section, we describe the recruitment process with BPMN diagrams. First, we use BPMN diagrams to model this process in an artifact-centric manner. After analyzing the recruitment process, one can find four artifacts involved in the process, i.e., person, employee, application and position. A BPMN diagram can be used to model the sub-process of one artifact.

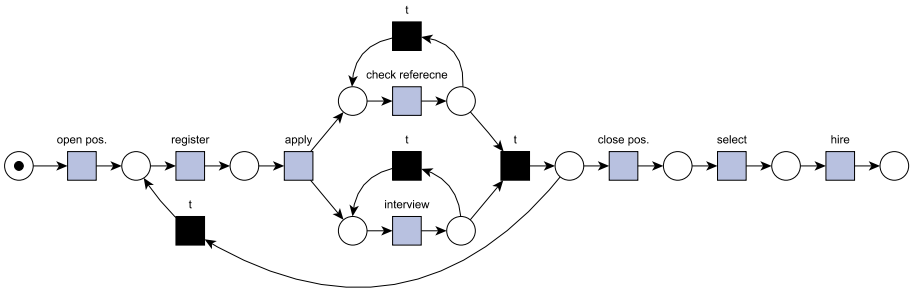


Figure 5.22: A Workflow net modeling the recruitment process selecting “position” as the case notion.

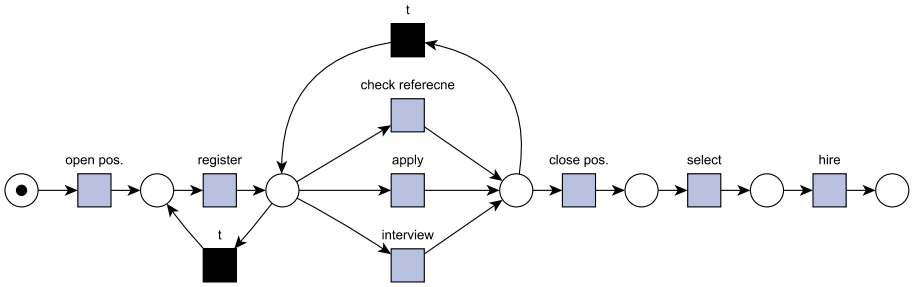


Figure 5.23: Another Workflow net modeling the recruitment process selecting “position” as the case notion.

Based on this idea, the whole process is split into four sub-processes. Figure 5.24 shows four BPMN models which describe the lifecycles of four artifacts (person, position, application, and employee) in separate diagrams. Compared with previous models, such as OCBC models, Declare models and Workflow nets, the BPMN models are very easy to understand. After identifying the artifacts/sub-processes based on the domain knowledge, each BPMN model describes a sub-process (the lifecycle of an artifact) clearly. However, these BPMN models fail to describe the one-to-many and many-to-many relationships between different artifacts. For instance, the one-to-many relationship between positions and applications is not shown at all in these BPMN models. In comparison, it can be clearly seen in the OCBC model in Figure 5.19 (i.e., the class

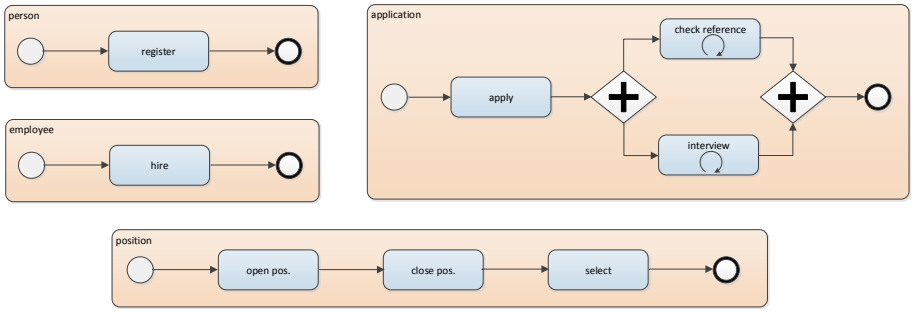


Figure 5.24: Modeling the recruitment process in terms of four BPMN models, each of which describe the lifecycle of an artifact.

relationship *r2* between class "application" and class "position"). The Workflow net in Figure 5.22 can also indicate the relationship through the loop.

Besides, the BPMN models in Figure 5.24 can not capture dependencies between different sub-processes. Consider for example constraints *con5*, *con6*, *con7*, and *con8* in the OCBC model of Figure 5.19, which can not be described by the BPMN models. More precisely, the BPMN models do *not* show that (i) one can only apply if the corresponding position is opened; (ii) one can only apply if the corresponding position is not yet closed; (iii) the person to be hired should have registered, applied, and had at least one interview; (iv) employees are hired after the completion of the selection process.

In order to model the interactions between different sub-processes, it is possible to integrate all sub-processes into one single diagram by selecting a particular case notion. Figure 5.25 describes the whole process in one BPMN model, in which "position" is selected as the case notion and each sub-process is described in a lane. We use the "Parallel Multiple Instance" sub-processes to describe the one-to-many relations between "position" and "register", and between "position" and "application". However, the parallel and "one-to-many" relations between "register" and "application" is missing.

Comparing the BPMN models in Figure 5.24 with the OCBC model in Figure 5.19 reveals that modeling the lifecycles of entities separately, like in artifact-centric approaches, is not sufficient to capture the real process. The individual lifecycles are simple, but fail to reveal the interplay between persons, positions, applications, and employees. The relations with the overall data model and interactions between the different entities are no longer visible. Figure 5.25 can

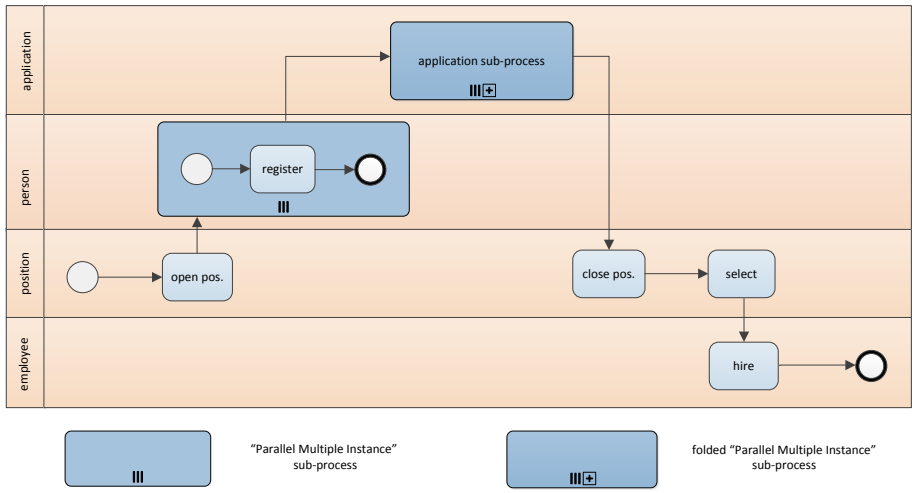


Figure 5.25: Modeling the recruitment process in terms of one BPMN model with four lanes.

describe the interplay between entities by using lanes and "Parallel Multiple Instance" sub-processes. However, it depends on a case notion and the interactions between some entities are missing.

5.5.5 Discussion of Evaluation

Table 5.4 summarizes the evaluation results explained above. The WF-net (data-aware) refers to the complete net, i.e., with the data perspective, in Figure 5.21 while the WF-net (only control-flow) refers to the control-flow of the same net. The BPMN diagram (artifact-centric) corresponds to Figure 5.24 and the BPMN diagram (case-centric) corresponds to Figure 5.25.

The expressive ability means the ability to describe the business process, which is discussed from behavioral and data perspectives as well as the interaction between these two perspectives. The two WF-nets and the BPMN diagram have strong ability to describe the behavioral perspective, since they use the procedural manner to model the lifecycle of a selected case. In contrast, the OCBC model and the Declare model use the declarative manner to model the constraints between pairs of activities. The BPMN diagram (artifact-centric) de-

language	expressive ability			complexity			hierarchy	multiple instances
	behavioral	data	interaction	#nodes	#edges	degree		
OCBC model	weak	strong	strong	12	30	high	No	strong
Declare model	weak	-	-	8	9	low	No	weak
WF-net (data-aware)	strong	weak	weak	27	34	high	No	weak
WF-net (only control-flow)	strong	-	-	23	26	medium	No	weak
BPMN diagram (artifact-centric)	weak	weak	weak	18	15	low	No	weak
BPMN diagram (case-centric)	strong	weak	weak	18	16	low	Yes	weak

Table 5.3: The evaluation results.

scribes the process in terms of four artifacts, resulting in separate sub-processes, which do not provide a complete view of the process. In terms of the other two angles, the OCBC model is the only one having the strong power to describe the data perspective and interactions. The Declare model and WF-net (only control-flow) do not have the data perspective nor interactions. WF-net (data-aware), BPMN diagram (artifact-centric) and BPMN diagram (case-centric) use business objects, artifacts and lanes to describe the data perspective and interactions, in a weak manner compared with the OCBC model.

Due to the extra data perspective, the OCBC model and the WF-net (data-aware) have a higher complexity than others. In these models, the BPMN diagram (case-centric) uses a hierarchy to simplify the process on the top level. With the powerful data perspective, the OCBC model can better describe multiple instances than others.

5.6 Related Work

This chapter proposed a modeling language to describe the processes on artifact-centric information systems. It can solve the key problems faced by mainstream process modeling notations, summarized as follows:

- it is difficult to identify a case notion for the whole process;
- it is difficult to model interactions (one-to-many and many-to-many relationships) between process instances;
- it is difficult to model the data-perspective and behavioral (control-flow) perspective in a unified and integrated manner.

There are attempts to solve the problems mentioned above (see for example [154]), but they fail to provide a satisfactory solution. To position the work in existing literature on business process modeling, we discuss the related work in this section.

5.6.1 Process-Centric Notations Extended with Data Perspective

Over time there have been numerous attempts to take data into consideration by adding elements onto the behavioral perspective.

Consider for example the various types of *colored Petri nets*, i.e., Petri nets where tokens have attributes and values [46, 47, 72, 74, 163]. In data-aware Petri nets (DPNs) [33, 97, 127] and BPMN diagrams, activities can write/read data elements (variables and objects). In [131], the associations between services are modeled in a WFD-net (extending WorkFlow nets with Data), in which activities are annotated with the data that are created, read or written by the activities. The aforementioned approaches can describe the data perspective to some extent but do not support explicit data modeling as can be found in ER models [25], UML class models [53], and Object-Role Models (ORM) [60]. For instance, places and tokens in *colored Petri nets* can be typed, but there is no data model to relate entities and activities.

The first approaches that explicitly related process and data models appeared in the 1990-ties [154, 158]. For instance, the approach proposed by Kees van Hee [154] combined (i) Petri nets, extended with time, token values and hierarchy, (ii) a specification language that is a subset of Z, and (iii) a binary data model, extended with complex objects.

Some languages have the ability to represent the needed data, but their focus is on the sequencing of the activities that are carried out in the process. DFDs (data-flow diagrams) would be one example of this. While placing high importance on the data, the focus is on how these data move in the process, from one activity to next, and little importance is given to their details or on the structure of these data.

Object-aware process management, as supported by the *PHILharmonicFlows* framework [26, 75, 76], provides an integrated methodology for modeling requirements of process- and data-centric software systems. Key elements of the approach are the separation of concerns (data and process can be considered separately) and the ability to support change.

Other approaches integrating data and processes include *case handling systems* [150] and *product-based workflows* [156]. In fact, it is impossible to list all the different proposals described in literature.

To sum up, most process-centric languages focus on modeling the sequencing of the activities in business processes. They try to consider the data perspective by adding data elements, such as attributes, values, variables and objects, onto the control-flow. However, the ability on the data perspective of these approaches

is quite limited, e.g., only reading and writing variables. In consequence, it lacks the richness of more powerful notations used in data modeling language (e.g., a UML class model).

The OCBC language proposed in this chapter is different from the above approaches, because the data-perspective and control-flow perspective are modeled in a single diagram and a single unifying mechanism, namely *cardinality constraints*, is used to denote all types of constraints.

5.6.2 Artifact-Centric Approaches

Artifact-centric approaches [28,66,92,107] (including the earlier work on proclets [140]) aim to capture business processes in terms of so-called business artifacts, i.e., key entities which drive a company's operations and whose lifecycles and interactions define an overall business process.

Note that the artifact-centric approaches do not limit to a specific graphical notation. In Section 2.4.4, we introduced proclets [139, 140] and GSM models [66, 67] to model each artifact, including the lifecycle and attributes of each artifact. In this section, we compare artifacts and OCBC models.

Artifacts have data and lifecycles attached to them, thus relating both perspectives. However, the relationships between artifacts are not made explicit. In summary, these approaches tend to result in models where:

- the description of the end-to-end behavior needs to be distributed over multiple diagrams (e.g., one process model per artifact),
- the control-flow cannot be related to an overall data model (i.e., there is no explicit data model or it is separated from the control-flow), and
- interactions between different entities are not visible or separated (because artifacts are distributed over multiple diagrams).

Besides, artifacts have to be identified beforehand based on domain knowledge and within an artifact (proclet, or subprocess), *one is forced to pick a single instance notion*. Moreover, cardinality constraints in the data model cannot be exploited while specifying the intended dynamic behavior.

In comparison, OCBC models integrate the data and process perspectives better. More precisely, these two perspectives are combined in one single diagram, such that we can see (i) the end-to-end process, (ii) the structure of the entities involved in the process, and (iii) all interactions between different entities and perspectives. Besides, due to the employment of the unified cardinality constraints, the interactions between different parts in OCBC models are described in a consistent manner.

language	behavioral perspective		data perspective		interactions		whole	
	case notion	flexible	attributes	structure	vertical	horizontal	unified	single
ER	-	-	Yes	Yes	-	strong	-	-
UML class model	-	-	Yes	Yes	-	strong	-	-
Petri net	Yes	weak	-	-	-	weak	-	-
Declare	Yes	strong	-	-	-	strong	-	-
Colored Petri net	Yes	weak	Yes	No	weak	weak	No	Yes
BPMN	Yes	weak	Yes	No	weak	weak	No	No
Artifact-centric	No	strong	Yes	No	strong	strong	No	No
OCBC	No	strong	No	Yes	strong	strong	Yes	Yes

Table 5.4: Modeling language examples.

Table 5.4 shows a comparison of different modeling languages. In general, we split the comparison into four dimensions, i.e., the behavioral perspective, the data perspective, the interactions and the model as a whole. More precisely, on the behavioral perspective, we check if these languages need a case notion (Yes/No) and compare their abilities to deal with flexible processes (strong/weak). The ER models and UML class models cannot describe the behavioral perspective, indicated by the value "-". Artifact-centric approaches and OCBC models do not require a case notion and Declare models and OCBC models can deal with flexible processes better. On the data perspective, only ER models, UML class models and OCBC models are able to describe the structure of entities involved in business processes while others can only show the attributes (representing variables, values and objects). Note that currently we do not add attributes to the classes in OCBC models, but they could be added in a straightforward manner in the future work. In terms of the interactions, we define two types, the "vertical" interactions between data and behavioral perspectives and the "horizontal" interactions between different instance types. Both artifact-centric approaches and OCBC models have strong power to describe these two types of interactions. For the last dimension, OCBC models are outstanding since they employ the unified cardinality constraints and an OCBC model can be represented by a single diagram.

5.7 Summary

In this chapter, we proposed *Object-Centric Behavioral Constraint* (OCBC) models as a means to graphically model control-flow and data/objects in an integrated manner. Cardinality constraints are used to *specify structure and behavior in*

a single diagram. In existing approaches, there is often a complete separation between data/structure (e.g., a class model) and behavior (e.g., BPMN, EPCs, or Petri nets). In OCBC models, different types of instances can interact in a fine-grained manner and the constraints in the class model guide behavior.

This work serves as a starting point for a new line of research. A discovery technique to transform an XOC log into an OCBC model is proposed in Chapter 6. OCBC models are particularly suitable for conformance checking. Many deviations can only be detected by considering multiple instances and constraints in the class model. In Chapter 7, we identify nine types of conformance problems that can be detected using OCBC models. Besides, the performance analysis based on OCBC models is illustrated in Chapter 8, which can show the bottlenecks of the business processes.

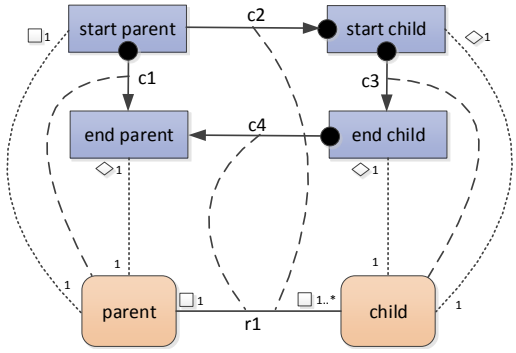


Figure 5.26: Example pattern. After starting the parent, a particular number of children k where $1 \leq k$ (as defined by $r1$) need to start. After all k children ended, the parent ends.

In fact, there are many possible avenues for *future work*. We also want to identify typical behavioral *patterns* that involve multiple instances or interaction between structure and behavior. Figure 5.26 shows an example pattern. Along this line, we plan to study the effect of introducing *subtyping* in the data model, a constraint present in all data modeling approaches. The interplay between behavioral constraints and subtyping can motivate other interesting behavioral patterns. For example, *implicit choices* may be introduced through subtyping. Consider a response constraint pointing to a *payment* class with two subclasses *credit card payment* and *cash payment*. Whenever the response constraint is activated and a payment is expected, such an obligation can be fulfilled by either

paying via cash or credit card.

Finally, we also want to investigate how the notions of *consistency* and *constraint conflict/redundancy*, well-known in the context of Declare [143], and the corresponding notions of *consistency* and *class consistency*, well-known in data models [11], can be suitably reconstructed and combined in our setting.

Chapter 6

Discovery of Object-Centric Behavioral Constraint Models

Model discovery is the first challenging task in process mining. Up to now, we have formed a foundation to apply model discovery to artifact-centric information systems. More precisely, Chapter 4 defined the XOC event log format and proposed an approach to extract such logs from the object-centric event data (generated by artifact-centric information systems) defined in Chapter 3. Since conventional models (such as Petri nets, BPMN diagrams) fail to deal with the processes on artifact-centric information systems properly, Chapter 5 introduced a novel modeling language, Object-Centric Behavioral Constraint (OCBC) modeling language, which can better describe these processes, resulting in OCBC models.

In this chapter, we discover OCBC models which are able to describe processes involving *interacting instances* and *complex data dependencies*. In order to enable the discovery approach working in real-life data, we also try to propose approaches which can deal with noise, i.e., discovering precise and understandable OCBC models from logs with infrequent events and objects.

This chapter is organized as follows. An initial approach is illustrated in Section 6.1, which contains the discovery of class models, AOC relationships and activity models. Section 6.2 proposes more advanced discovery approaches which can deal with the noise in real-life data. In Section 6.3, we evaluate the model discovery approach by comparing it with other model discovery approaches. Section 6.4 reviews the related work while Section 6.5 concludes this chapter.

6.1 An Initial Approach for OCBC Discovery

In this section, we illustrate an initial approach to discover OCBC models from logs extracted from artifact-centric systems. We first discuss the input of the approach and then illustrate the approach through three steps. More precisely, the first step is to discover a class model, which is the backbone of the whole OCBC model. Secondly, AOC relationships are discovered to build a bridge leading to the behavioral perspective from the data perspective (i.e., the class model). At last, we correlate events based on correlation patterns extracted from the data perspective and AOC relationships, and discover the activity model to finish the discovery of a whole OCBC model.

This approach serves as a foundation to understand the process of OCBC model discovery. Note that it is only an initial approach, which does not consider the noise yet, e.g., infrequent events or objects. In other words, it requires "clean" logs and discovers constraints in a naive way. In Section 6.2, we will introduce some more advanced approaches to deal with noise in real-life data.

6.1.1 Input for OCBC Discovery

A general model discovery approach can be considered as a function that maps an event log L onto a model M such that the model is "representative" for the business process which generated the event log. In other words, the input log decides the type and quality of the discovered model. In traditional process mining, the input for model discovery is often an XES log (it is possible to use other sources, such as CSV file and database tables). A main feature of the XES log is that it assumes a case notion and focuses more on the behavioral (control-flow) perspective. Accordingly, the discovered model from the log also has a case notion and mainly presents the behavioral perspective.

OCBC models are powerful to describe the processes on artifact-centric systems, since they (i) do not assume a case notion, (ii) contain data and behavioral perspectives, and (iii) correlate events by the data perspective. Therefore, we choose OCBC models as the output of our discovery approaches. Obviously, XES logs cannot support the discovery of OCBC models, since they are flattened (i.e., assuming case notions) and biased (i.e., focusing more on the behavioral perspective). In contrast, XOC logs satisfy the requirements to discover OCBC models, since in these logs, (i) data are not flattened by case notions (an XOC log directly consists of a list of events), (ii) each event corresponds to an object model which contains information on both data and behavioral perspectives, and (iii) each event refers to some objects, which provide "bridges" (i.e., objects) to

correlate events. Therefore, the discovery approaches take XOC logs as input to discover OCBC models.

Table 6.1 presents an XOC log $L = (E, act, attrE, relate, om, \preceq)$ which corresponds to the nine tables in the OTC scenario in Chapter 3. More precisely, the numbers in the "Index" column indicate the orders between events (i.e., $\preceq$). The "Event" column contains all events in the log (i.e., E), and an event is represented by an ID, e.g., $co1$ refers to the first event in the log. Each event has a corresponding activity indicated by the "Activity" column (i.e., *act*), e.g., the activity of $co1$ is co (the short name of "create order" as shown in Table 6.3). In order to connect the behavioral and the data perspectives (i.e., events and objects), each event refers to at least one object indicated by the "Reference" column (i.e., *relate*), e.g., $ci1$ refers to $i1$ and $er1$. Moreover, each event has a corresponding object model (i.e., *om*) consisting of objects (the "Objects" column) and object relations (the "Relations" column) in the "Object Model" column. For instance, the first event $co1$ has a corresponding object model, which contains the set of objects $\{c1, c2, o1, ol1, ol2\}$ and the set of object relations $\{(r5, c1, o1), (r10, o1, ol1), (r10, o1, ol2)\}$.

Note that each object has a corresponding class. In Table 6.1, the class of an object is indicated by the letters in its ID for simplicity. For instance, $ol1$ indicates that its corresponding class is ol (the short name of "order line" as shown in Table 6.2). Figure 6.1 presents the log in an evolutionary view. After the occurrence of some event, objects or object relations may be added, updated or deleted.

The definition of XOC logs $L = (E, act, attrE, relate, om, \preceq)$ is quite general (cf. Chapter 4). In the context of real-life data from relational databases (which is the scope for input data of our research), some logs are not possible although they are legal according to the definition. For instance, an event can refer to some objects in the object universe based on the definition. However, in real applications, an event can only refer to some existent objects, since an event (e.g., add, update or deletion) can only operate existent objects, i.e., records in the database. Therefore, we limit the input for our discovery approach to *sound logs* which can be derived from real-life data. By doing this, we make the discovered model interpretable in real applications.

An XOC event log is sound if it satisfies three rules explained next. In a sound log, each event can only refer to objects which exist in its corresponding object model or its prior object model (the first rule). This is consistent with real facts in databases. For instance, an operation (e) can add a record (o) and the record should exist after the operation, i.e., o exists in the object model corresponding to e. In a different scenario, an operation (e) can also delete a record (o) and

Figure 6.1: The evolutionary view of the XOC log example in Table 6.1. Note that an object model can be considered as the combination of its previous object model and some new elements. This can be used as a trick to easily understand the log.

Index	Event	Activity	References	Object Model	
				Objects	Relations
1	co1	co	o1,ol1,ol2	c1,c2,o1,ol1,ol2	(r5,c1,o1), (r10,o1,ol1), (r10,o1,ol2)
2	co2	co	o2,ol3,ol4	c1,c2,o1,ol1,ol2, o2,ol3,ol4	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4)
3	cs1	cs	s1,ol1	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1)
4	ci1	ci	i1,er1	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1), (r8,o1,er1)
5	cs2	cs	s2,ol1,ol2	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1), (r8,o1,er1), (r4,c1,s2), (r6,s2,sl2), (r9,ol1,sl2), (r6,s2,sl3), (r9,ol2,sl3)
6	ci2	ci	i2,er2,er3	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3, i2,er2,er3	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1), (r8,o1,er1), (r4,c1,s2), (r6,s2,sl2), (r9,ol1,sl2), (r6,s2,sl3), (r9,ol2,sl3), (r1,c1,i2), (r3,i2,er2), (r8,o1,er2), (r3,i2,er3), (r8,o2,er3)
7	cs3	cs	s3,ol3,ol4	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3, i2,er2,er3,s3,sl4, sl5	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1), (r6,s1,sl1), (r9,ol1,sl1), (r1,c1,i1), (r3,i1,er1), (r8,o1,er1), (r4,c1,s2), (r6,s2,sl2), (r9,ol1,sl2), (r6,s2,sl3), (r9,ol2,sl3), (r1,c1,i2), (r3,i2,er2), (r8,o1,er2), (r3,i2,er3), (r8,o2,er3), (r4,c1,s3), (r6,s3,sl4), (r9,ol3,sl4), (r6,s3,sl5), (r9,ol4,sl5)
8	cp1	cp	p1,pl1,pl2	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3, i2,er2,er3,s3,sl4, sl5,p1,pl1,pl2	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4),(r4,c1,s1),(r6,s1,sl1),(r9,ol1,sl1),(r1,c1,i1), (r3,i1,er1),(r8,o1,er1),(r4,c1,s2),(r6,s2,sl2),(r9,ol1,sl2), (r6,s2,sl3),(r9,ol2,sl3),(r1,c1,i2),(r3,i2,er2),(r8,o1,er2), (r3,i2,er3),(r8,o2,er3),(r4,c1,s3),(r6,s3,sl4),(r9,ol3,sl4), (r6,s3,sl5),(r9,ol4,sl5),(r7,p1,pl1),(r2,i1,pl1),(r7,p1,pl2), (r2,i2,pl2)
9	ci3	ci	i3,er4	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3, i2,er2,er3,s3,sl4, sl5,p1,pl1,pl2,i3, er4	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4), (r4,c1,s1),(r6,s1,sl1),(r9,ol1,sl1),(r1,c1,i1), (r3,i1,er1),(r8,o1,er1),(r4,c1,s2),(r6,s2,sl2),(r9,ol1,sl2), (r6,s2,sl3),(r9,ol2,sl3),(r1,c1,i2),(r3,i2,er2),(r8,o1,er2), (r3,i2,er3),(r8,o2,er3),(r4,c1,s3),(r6,s3,sl4),(r9,ol3,sl4), (r6,s3,sl5),(r9,ol4,sl5),(r7,p1,pl1),(r2,i1,pl1),(r7,p1,pl2), (r2,i2,pl2),(r1,c1,i3),(r3,i3,er4),(r8,o2,er4)
10	cp2	cp	p2,pl3	c1,c2,o1,ol1,ol2, o2,ol3,ol4,s1,sl1, i1,er1,s2,sl2,sl3, i2,er2,er3,s3,sl4, sl5,p1,pl1,pl2,i3, er4,p2,pl3	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2),(r10,o2,ol3), (r10,o2,ol4),(r4,c1,s1),(r6,s1,sl1),(r9,ol1,sl1),(r1,c1,i1), (r3,i1,er1),(r8,o1,er1),(r4,c1,s2),(r6,s2,sl2),(r9,ol1,sl2), (r6,s2,sl3),(r9,ol2,sl3),(r1,c1,i2),(r3,i2,er2),(r8,o1,er2), (r3,i2,er3),(r8,o2,er3),(r4,c1,s3),(r6,s3,sl4),(r9,ol3,sl4), (r6,s3,sl5),(r9,ol4,sl5),(r7,p1,pl1),(r2,i1,pl1),(r7,p1,pl2), (r2,i2,pl2),(r1,c1,i3),(r3,i3,er4),(r8,o2,er4),(r7,p2,pl3), (r2,i3,pl3)

Table 6.1: An XOC log example as the input for discovery. Note that an object model can be considered as the combination of its previous object model and some new elements. This can be used as a trick to easily understand the log.

Table 6.2: Classes and corresponding objects

Class	Short name	Objects
order	o	o1,o2
order line	ol	ol1,ol2,ol3,ol4
shipment	s	s1,s2,s3
shipment line	sl	sl1,sl2,sl3,sl4,sl5
invoice	i	i1,i2,i3
payment	p	p1,p2
payment line	pl	pl1,pl2,pl3
customer	c	c1,c2
element relation	er	er1,er2,er3,er4

Table 6.3: Activities and corresponding events

Activity	Short name	Events
create order	co	co1,co2
create shipment	cs	cs1,cs2,cs3
create invoice	ci	ci1,ci2,ci3
create payment	cp	cp1,cp2

the record should exist just before the operation, i.e., o exists in the object model corresponding to the event just before e. The second rule requires that each object should always have the same class, which conforms to the approach of extracting XOC logs from databases. In the process of extracting logs, each object corresponds to precisely one record and the class of the object corresponds to the table where the record locates. If an object has different classes, it means that the object corresponds to multiple records in different tables, which violates the approach and is impossible in the extracted logs. The last rule indicates that object relations of the same class relationship should have the same type of source (first) and target (second) objects.[1] This rule can be explained in the context of databases, where (i) a class relationship corresponds to some FK-PK relationship, and the object relations of the class relationship correspond to the instantiations of the FK-PK relationship, and (ii) the instantiations can only be references from records in the child table (including FK) to records in the father table (including PK). Based on the rules discussed above, we define the sound XOC event log.

Definition 6.1 (Sound XOC Event Log) *An XOC* event log $L = (E, act, attrE, relate, om, \preceq)$ *is sound if and only if*

- $\forall e \in E, o \in relate(e) : \exists e' \in \trianglelefteq^2_e(E) : o \in Obj_{e'}$*, i.e., the objects referred to by an event e exist in the object model corresponding to e or to the event prior to e (i.e., the closet event before e),* [2]
- $\forall e \in E, o \in Obj_e, e' \in E, o' \in Obj_{e'} : o = o' \Rightarrow class_e(o) = class_{e'}(o')$*, i.e., objects*

[1] In an object relation (r, o_1, o_2), r indicates the type of the relation, $o1$ is the source object and $o2$ is the target object.

[2] $\trianglelefteq^n_e E$ gets the closest n events before or equal to e.

cannot change classes in-between events,[3]

- $\forall e, e' \in E : \forall (r, o_1, o_2) \in Rel_e, (r', o_1', o_2') \in Rel_{e'} : r = r' \Rightarrow class_e(o_1) = class_{e'}(o_1')$ $\wedge\, class_e(o_2) = class_{e'}(o_2')$*, i.e., all the first (second) objects of object relations corresponding to the same relationship are of the same class.*

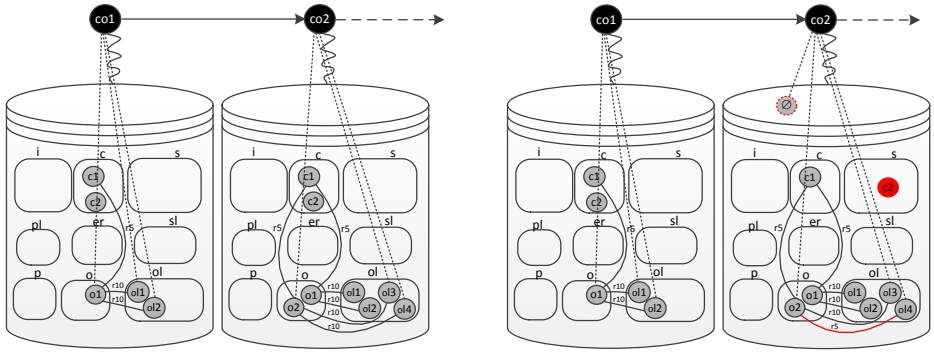


Figure 6.2: A comparison of sound and unsound event logs.

Figure 6.2 shows a comparison of a sound log and an unsound log to understand the rules defined above. Figure 6.2(a) zooms in on the first two events ($co1$ and $co2$) of the log in Figure 6.1. The object models contain objects of three classes, such as $c1$ and $c2$ of c (denoting "customer" class), $o1$ and $o2$ of o (denoting "order" class) and $ol1$, $ol2$, $ol3$ and $ol4$ of ol (denoting "order line" class). There exist object relations of $r5$, e.g., $(r5, c1, o1)$, between c objects and o objects, and object relations of $r10$, e.g., $(r10, o1, ol1)$, between o objects and ol objects. Based on the sound log, we modify some objects and object relations, resulting in the unsound log in Figure 6.2(b).

In Figure 6.2(b), the parts violating the rules are highlighted in red. More precisely, the event $co2$ refers to an object $\emptyset$, which does not exist in the object model corresponding to $co2$ or $co1$. Besides, in the first object model, object $c2$ is of class c and its class changes to s in the second object model. Moreover,

[3]This requirement is reasonable since classes in this thesis are disjoint and do not contain subclasses. Note that this may not be the case anymore if subclasses exist and objects can evolve across time. For instance, assume that class "order" has two subclasses "open order" and "closed order". An object of class "open order" at some point in time may become a "closed order" in the future.

the object relations of $r5$ connect c objects to o objects in the first object model. However, there is an object relation of $r5$ connecting an o object to an ol object.

The log in Table 6.1 is a sound log, since it conforms to all the rules. It will be used as the input to illustrate the discovery approach in the next section.

6.1.2 Discovery of Class Models

An OCBC model consists of a class model, an activity model and AOC relationships which connect these two models. Note that OCBC models are data/object-centric, i.e., the data perspective is the backbone and events on the behavioral perspective are correlated through the data perspective. Therefore, we first show how to discover a *class model* which represents the data perspective.

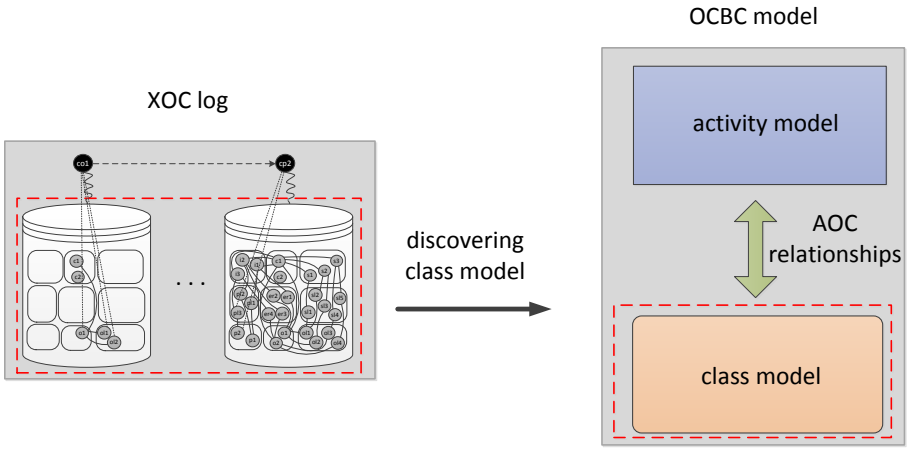


Figure 6.3: A class model is discovered based on all object models in an XOC log.

From the modeling angle, a class model defines a "space" of possible object models. On the contrary, observed object models specify a class model from the discovery angle. In other words, the class model can be discovered based on all object models in an XOC log as shown in Figure 6.3. When discovering a class model $ClaM = (C, R, \pi_1, \pi_2, \sharp^{\square}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\square}_{tar}, \sharp^{\Diamond}_{tar})$, the discovery of cardinality functions (i.e., $\sharp^{\square}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\square}_{tar}$ and $\sharp^{\Diamond}_{tar}$) is the core task, and it depends on the result of object correlation. Therefore, we next illustrate how to correlate objects by

class relationships and identify related objects of a given object, which serves as the base for discovering a class model.

Definition 6.2 (Objects Observed in Log) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log. O_L is the set of all objects in L, i.e., $O_L = \{o \mid \exists e \in E : o \in Obj_e\}$. For an event $e \in E$ and an object $o \in O_L \setminus Obj_e$, $class_e(o) = class_{e'}(o)$ where $e' \in E$ and $o \in Obj'_e$.*

O_L consists of all objects in a log L, which have been observed in at least one object model corresponding to some event. Note that it is possible that an object $o \in O_L$ appears in an object model and is deleted later. For an event $e \in E$, function $class_e$ is defined (cf. Chapter 3) to give the class of an object in the object model of event e. Since the class of an object stays the same in all object models, we can use $class_e$ corresponding to any object model to derive the class. If an object o does not exist in the object model (i.e., $o \in O_L \setminus Obj_e$), function $class_e$ returns the class of o in another object model which includes o (i.e., $class_e(o) = class_{e'}(o)$ where $e' \in E$ and $o \in Obj'_e$).

Definition 6.3 (Relating Objects by Class Relationship) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log, O_L be the set of all objects, and R be a set of class relationships. Function $corR \in O_L \times R \times E \rightarrow \mathbb{P}(O_L)$ correlates objects by a class relationship $r \in R$ and returns all objects related to a given object $o \in O_L$ at some moment that event $e \in E$ happens, such that*

- *if o is an object of the* source *class of r, i.e., $o \in \partial_{\pi_1(r)}(O_L)$,*

$$corR_{src}(o, r, e) = \{o' \mid (r, o, o') \in Rel_e\},$$

- *if o is an object of the* target *class of r, i.e., $o \in \partial_{\pi_2(r)}(O_L)$,*

$$corR_{tar}(o, r, e) = \{o' \mid (r, o', o) \in Rel_e\},$$

where $corR_{src}$ and $corR_{tar}$ are two variants of corR.

Given an object o and a class relationship r, function $corR$ correlates objects by r and returns all objects related to o. Note that the object model is evolving, i.e., the related objects of o may change over time. It is necessary to indicate the time when identifying the related objects. Here, the time is represented by an event, which means that the time is when the event happens. For each object

of the *source* class of r, i.e., $o \in \partial_{\pi_1(r)}(O_L)$, an object o' is related to o, if (i) o' is connected to o through an object relation of r and (ii) o' is the *target* object of the object relation, i.e., $(r, o, o') \in Rel_e$. In contrast, if an object o is of the *target* class of r, o should be the target object of the object relation (i.e., $(r, o', o) \in Rel_e$) when identifying its related objects.

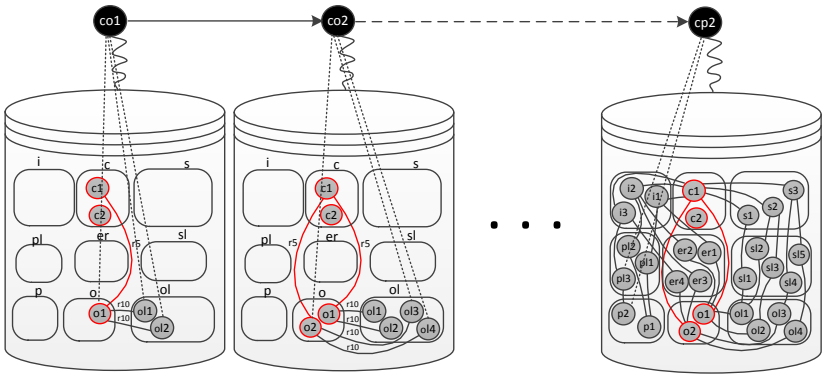


Figure 6.4: Correlating objects by a class relationship $r5$.

Consider the log in Table 6.1 and a class relationship $r5$ as an example to understand the object correlation, where $\pi_1(r5) = customer$ and $\pi_2(r5) = order$. As shown in Figure 6.4, $c1$ and $c2$ are objects of the source class of $r5$ and $o1$ and $o2$ are objects of the target class of $r5$, which are highlighted in red. In the object model corresponding to event $co1$, since only $o1$ is connected to $c1$ by an object relation $(r5, c1, o1)$, we get $corR_{src}(c1, r5, co1) = \{o1\}$. Similarly, $corR_{src}(c1, r5, co2) = \{o1, o2\}$ since $o1$ and $o2$ are connected to $c1$ in the object model corresponding to $co2$. In this way, we compute the related objects of $o1$, $o2$, $c1$ and $c2$ at each moment and the result is shown in Table 6.4. Note that it is possible that an object does not exist at some moment. In this situation, the related objects of the inexistent object are denoted by "-", e.g., $o2$ does not exist in the object model corresponding to event $co1$.

The discovery of the class model can be done after correlating objects. More precisely, the set of classes C is derived by integrating the classes of all objects in object models. The set of class relationships R and corresponding functions π_1 and π_2 are based on the object relations (i.e., types, source objects and target objects in object relations, respectively). The cardinality functions of a class relationship can be derived based on correlated objects. For "always" cardinality

object	related objects $corR(o, r5, e_i)$									
	$co1$	$co2$	$cs1$	$ci1$	$cs2$	$ci2$	$cs3$	$cp1$	$ci3$	$cp2$
$o1$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$
$o2$	-	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$	$\{c1\}$
$c1$	$\{o1\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$	$\{o1, o2\}$
$c2$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 6.4: Related objects (correlated by $r5$) of each *order* or *customer* object in each object model (represented by the corresponding event ID) in Table 6.1.

functions ($\sharp^{\Box}_{src}$ and $\sharp^{\Box}_{tar}$), we need to check each object model (i.e., each moment) and for "eventually" cardinality functions ($\sharp^{\Diamond}_{src}$ and $\sharp^{\Diamond}_{tar}$) we only check the last object model (i.e., the last moment).

Definition 6.4 (Discovery of Class Models) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC log. The discovered class model from L is a tuple $ClaM = (C, R, \pi_1, \pi_2, \sharp^{\Box}_{src}, \sharp^{\Diamond}_{src}, \sharp^{\Box}_{tar}, \sharp^{\Diamond}_{tar})$, where*

- *C is a set of object classes such that $C = \{c \mid \exists e \in E, o \in Obj_e : class_e(o) = c\}$,*
- *R is a set of relationships such that $R = \{r \mid \exists e \in E : (r, o_1, o_2) \in Rel_e\}$,*
- *$\pi_1 \in R \rightarrow C$ gives the source class of a relationship such that $\forall r \in R : \pi_1(r) = c_1$ where $\forall e \in E : \forall (r, o_1, o_2) \in Rel_e : c_1 = class_e(o_1)$,*
- *$\pi_2 \in R \rightarrow C$ gives the target class of a relationship such that $\forall r \in R : \pi_2(r) = c_2$ where $\forall e \in E : \forall (r, o_1, o_2) \in Rel_e : c_2 = class_e(o_2)$,*
- *$\sharp^{\Box}_{src} \in R \rightarrow \mathcal{U}_{Card}$ gives the source "always" cardinality of a relationship such that $\forall r \in R : \sharp^{\Box}_{src}(r) = \{n \mid \exists e \in E, o_2 \in \partial_{\pi_2(r)}(Obj_e) : n = |corR_{tar}(o_2, r, e)|\}$,*
- *$\sharp^{\Diamond}_{src} \in R \rightarrow \mathcal{U}_{Card}$ gives the source "eventually" cardinality of a relationship such that $\forall r \in R : \sharp^{\Diamond}_{src}(r) = \{n \mid \exists o_2 \in \partial_{\pi_2(r)}(Obj_{e_l}) : n = |corR_{tar}(o_2, r, e_l)|\}$, where e_l is the last event in L,*
- *$\sharp^{\Box}_{tar} \in R \rightarrow \mathcal{U}_{Card}$ gives the target "always" cardinality of a relationship such that $\forall r \in R : \sharp^{\Box}_{tar}(r) = \{n \mid \exists e \in E, o_1 \in \partial_{\pi_1(r)}(Obj_e) : n = |corR_{src}(o_1, r, e)|\}$, and*
- *$\sharp^{\Diamond}_{tar} \in R \rightarrow \mathcal{U}_{Card}$ gives the target "eventually" cardinality of a relationship such that $\forall r \in R : \sharp^{\Diamond}_{tar}(r) = \{n \mid \exists o_1 \in \partial_{\pi_1(r)}(Obj_{e_l}) : n = |corR_{src}(o_1, r, e_l)|\}$, where e_l is the last event in L.*

Figure 6.5 shows a discovered class model from the example log in Table 6.1 with the algorithm in Definition 6.4. The following steps illustrate the process of the discovery of the class model. C is learned by incorporating classes of all objects in each object model. For instance, *customer* is a discovered class since the object model of the event $co1$ contains an object $c1$ of class *customer*. In this way, nine classes are discovered, i.e., $C = \{customer, order, order\ line, shipment, shipment\ line, invoice, element\ relation, payment, payment\ line\}$. Similarly, R is derived by incorporating all relationships, i.e., types of object relations in each object model. For example, the relationship $r5$ is discovered because the object model of the event $co1$ contains an object relation $(r5, c1, o1)$ of $r5$. In this way, ten class relationships are discovered, i.e., $R = \{r1, r2, ..., r10\}$.

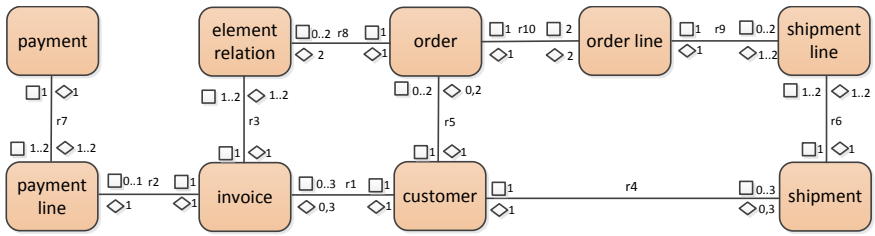


Figure 6.5: The discovered class model from the example log in Table 6.1.

Function π_1 (π_2) of each relationship r can be learned by observing the class of source (target) objects in all object relations of r. Consider for example the relationship $r5$. $\pi_1(r5) = customer$ and $\pi_2(r5) = order$ since there exists an object relation $(r5, c1, o1)$ (in the object model of the event $co1$), where the source object $c1$ is of class *customer* and the target object $o1$ is of class *order*. Note that it is enough to observe only one object relation of $r5$, because the source (target) objects in all object relations of the same relationship should have the same class in a sound log (cf. Definition 6.1).

For each relationship r, $\sharp^{\square}_{src}(r)$ ($\sharp^{\square}_{tar}(r)$) is specified as a set of integers, which integrates all the numbers of related objects (correlated by r) of each $\pi_2(r)$ ($\pi_1(r)$) object at each moment. Consider for example the target cardinality on relationship $r5$, i.e., $\sharp^{\square}_{tar}(r5)$. $\pi_1(r5)$ objects are *customer* objects $c1$ and $c2$, and we can know the number of related objects of each *customer* object at each moment according to Table 6.4. For instance, the object $c1$ ($c2$) has two (zero) related object at the last moment represented by event $cp2$. By integrating the numbers at all moments, $\sharp^{\square}_{tar}(r5) = \{0, 1, 2\}$. For $\sharp^{\diamondsuit}_{src}(r)$ and $\sharp^{\diamondsuit}_{tar}(r)$, we only need to integrate the numbers at the last moment. Therefore, $\sharp^{\diamondsuit}_{tar}(r5) = \{0, 2\}$.

6.1.3 Discovery of AOC Relationships

In Section 6.1.2, we illustrated how to discover the data perspective, i.e., a class model. The subsequent challenge is to discover the behavioral perspective, i.e., behavioral constraints between activities. Different from existing discovery approaches based on XES logs, events in XOC logs should be related through the data perspective before discovering behavioral constraints, due to the lack of case notions. AOC relationships relate the data and behavioral perspectives, i.e., they provide a bridge from the discovered data perspective to the yet undiscovered behavioral perspective. Therefore, we discover AOC relationships in this section to enable the discovery of the behavioral perspective in the next section.

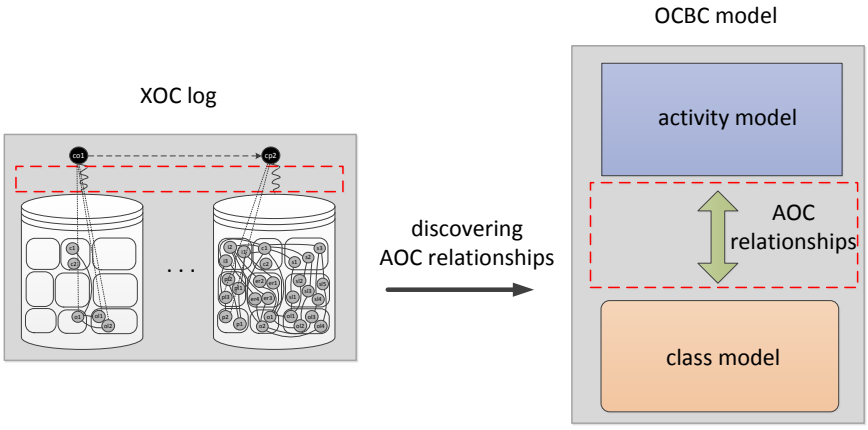


Figure 6.6: AOC relationships are discovered based on the references between events and objects.

From the modeling angle, AOC relationships describe the possible references between events and objects. On the other hand, the observed references indicate the AOC relationships from the discovery angle. In other words, the AOC relationships can be discovered based on all references in the XOC log (i.e., the "Reference" column in Table 6.1), as shown in Figure 6.6. When discovering AOC relationships, the discovery of cardinality functions (i.e., $\sharp_A^{\square}$, $\sharp_A^{\Diamond}$ and $\sharp_{OC}$) is the core task, and it depends on the correlation result involving events and objects. For an event e in an XOC log, its related objects are easily derived by $relate(e)$, and these related objects are constantly referred to by e from the moment that

e happens. In contrast, the events related to an object may change over time. For instance, at some point in time t, only event e refers to object o. At some moment later t', another event e' happens and also refers to o. In this situation, the related events of o are $\{e\}$ at t and $\{e, e'\}$ at t'. Therefore, an object may have different numbers of related events at different moments. It is necessary to indicate the time when identifying the related events of an object. Here, the time is represented by an event, which means the time is when the event happens.

Definition 6.5 (Correlating Events to Objects by AOC Relationship) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log, O_L be the set of all objects in L, and AOC be a set of AOC relationships. Function $corAOC \in O_L \times AOC \times E \rightarrow \mathbb{P}(E)$ correlates events to objects by an AOC relationship $(a, c) \in AOC$ and returns all events related to a given object $o \in \partial_c(O_L)$ at the moment that event $e \in E$ happens, such that*

$$corAOC(o, (a, c), e) = \{e' \in \trianglelefteq_e(E) \mid act(e') = a \wedge o \in relate(e')\}.$$

Given an object $o \in \partial_c(O_L)$ and an AOC relationship (a, c), function $corAOC$ correlates events to objects by (a, c) and returns all events related to o at some moment represented by an event e. More precisely, if an event e' (i) happens before or at e (i.e., $e' \in \trianglelefteq_e(E)$), (ii) corresponds to activity a (i.e., $act(e') = a$), and (iii) refers to o (i.e., $o \in relate(e')$), it is related to o at the moment e.

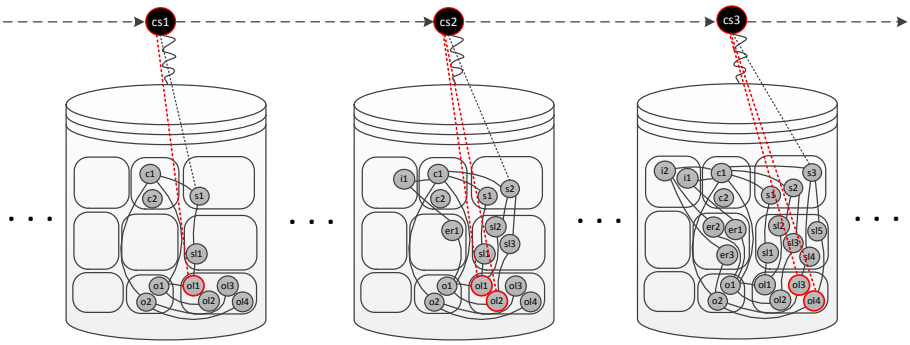


Figure 6.7: All *create shipment* events and corresponding object models in the example log in Table 6.1.

Consider the log in Table 6.1 and an AOC relationship $aoc7 = (create\ shipment, order\ line)$ as an example to understand the correlation. In Figure 6.7, we only

present *create shipment* events, i.e., $cs1$, $cs2$ and $cs3$, and corresponding object models. The involved events, objects and references (between events and objects) are highlighted in red. At the moment that event $cs1$ happens, since only $cs1$ refers to $ol1$, we get $corAOC(ol1, aoc7, cs1) = \{cs1\}$. In contrast, up to the moment that event $cs2$ happens, there are two events $cs1$ and $cs2$ referring to $ol1$, and therefore $corAOC(ol1, aoc7, cs2) = \{cs1, cs2\}$. In this way, we compute the related events of $ol1$, $ol2$, $ol3$ and $ol4$ at each moment and the result is shown in Table 6.5. Note that it is possible that an object does not exist at some moment. If an object does not exist, the set of its related events is denoted by "-", e.g., $ol3$ does not exist at the moment represented by event $co1$, corresponding to the "-" in the cell of the third row and the first column.

object	related events $corAOC(o, aoc7, e_i)$									
	$co1$	$co2$	$cs1$	$ci1$	$cs2$	$ci2$	$cs3$	$cp1$	$ci3$	$cp2$
$ol1$	$\emptyset$	$\emptyset$	$\{cs1\}$	$\{cs1\}$	$\{cs1, cs2\}$	$\{cs1, cs2\}$	$\{cs1, cs2\}$	$\{cs1, cs2\}$	$\{cs1, cs2\}$	$\{cs1, cs2\}$
$ol2$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{cs2\}$	$\{cs2\}$	$\{cs2\}$	$\{cs2\}$	$\{cs2\}$	$\{cs2\}$
$ol3$	-	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{cs3\}$	$\{cs3\}$	$\{cs3\}$	$\{cs3\}$
$ol4$	-	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{cs3\}$	$\{cs3\}$	$\{cs3\}$	$\{cs3\}$

Table 6.5: The events correlated to each *order line* object by $aoc7 = (create\ shipment, order\ line)$ at each moment (represented by an event) in the example log in Table 6.1.

Definition 6.6 (Discovery of AOC Relationships) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC log and O_L be the set of all objects in L. $AOC \subseteq \mathcal{U}_A \times \mathcal{U}_C$ is a set of relationships relating activities to classes discovered from L such that $AOC = \{(a, c) \mid \exists e \in E, o \in relate(e) : act(e) = a \wedge class_e(o) = c\}$. $\sharp_A^{\Box}$, $\sharp_A^{\Diamond}$ and $\sharp_{OC}$ are discovered cardinality functions for AOC where*

- *$\sharp_A^{\Box} \in AOC \rightarrow \mathcal{U}_{Card}$ gives the source "always" cardinality (on the activity side) of an AOC relationship such that $\forall (a, c) \in AOC : \sharp_A^{\Box}((a, c)) = \{n \mid \exists e \in E, o \in \partial_c(O_L) : n = |corAOC(o, (a, c), e)|\}$,*
- *$\sharp_A^{\Diamond} \in AOC \rightarrow \mathcal{U}_{Card}$ gives the source "eventually" cardinality (on the activity side) of an AOC relationship such that $\forall (a, c) \in AOC : \sharp_A^{\Diamond}((a, c)) = \{n \mid \exists o \in \partial_c(O_L) : n = |corAOC(o, (a, c), e_l)|\}$, where e_l is the last event in L, and*
- *$\sharp_{OC} \in AOC \rightarrow \mathcal{U}_{Card}$ gives the target cardinality (on the class side) of an AOC relationship such that $\forall (a, c) \in AOC : \sharp_{OC}(a, c) = \{n \mid \exists e \in \partial_a(E) : n = |\{o \in \partial_c(relate(e))\}|\}$*

Figure 6.8 shows the discovered AOC relationships and corresponding cardinalities (attached on the discovered class model) from the example log in

Table 6.1 using the algorithm in Definition 6.6. The following steps illustrate the process of the discovery of the AOC relationships.

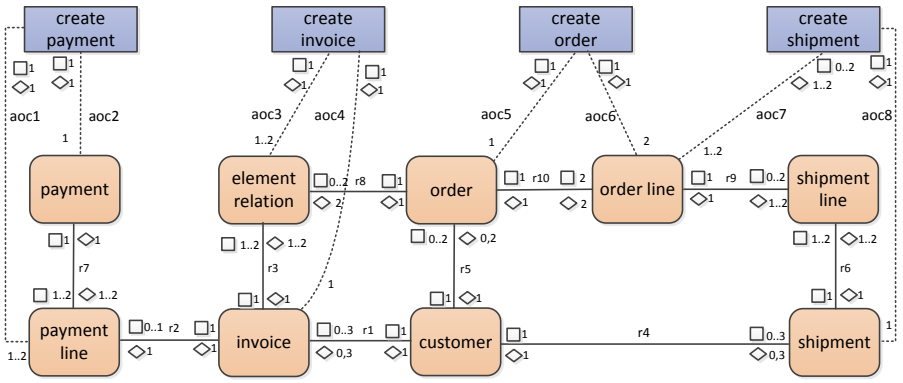


Figure 6.8: The discovered AOC relationships from the example log in Table 6.1.

First, we explain how to discover AOC relationships (without cardinality functions). If an event refers to an object, the activity of the event refers to the class of the object. For instance, since a *create shipment* event $cs1$ refers to one *order line* object $ol1$, as shown in the first object model in Figure 6.7, one AOC relationship (*create shipment*, *order line*) is discovered as $aoc7$ in Figure 6.8. In this way, we derive eight AOC relationships, i.e., $AOC = \{aoc1, aoc2, ..., aoc8\}$ in Figure 6.8.

Based on the discovered AOC relationships, cardinality functions are discovered based on the numbers of corresponding references between events and objects. For each AOC relationship (a, c), its target cardinality on the class side can be determined by incorporating numbers of references which relate c objects to each a event. Consider for example the target cardinality on $aoc7 = (\textit{create shipment}, \textit{order line})$. The numbers for *create shipment* events $cs1$, $cs2$ and $cs3$ are 1, 2 and 2, respectively, as shown in Figure 6.7. Therefore, $\sharp_{OC}(aoc7) = \{1,2\}$ (denoted as 1..2 in Figure 6.8).

Similarly, for each AOC relationship (a, c), the "always" cardinality on the activity side can be determined by incorporating numbers of related a events of each c object at every moment. We still use the AOC relationship $aoc7$ as an example. According to Table 6.5, the observed numbers at all moments for *order line* objects $ol1$, $ol2$, $ol3$ and $ol4$ are $\{0,1,2\}$, $\{0,1\}$, $\{0,1\}$ and $\{0,1\}$, respectively. By incorporating these numbers, $\sharp_A^{\Box}(aoc7) = \{0,1,2\}$. The "eventually"

cardinality on the activity side can be derived by incorporating numbers of related a events of each c object at the last moment, i.e., after all events happen. According to Table 6.5, the observed numbers at the last moment for *order line* objects $ol1$, $ol2$, $ol3$ and $ol4$ are 2, 1, 1 and 1, respectively. By incorporating these numbers, $\sharp_A^{\Diamond}(aoc7) = \{1,2\}$.

6.1.4 Discovery of Activity Models

Up to now, we have discovered the class model and AOC relationships. The activity model is the last piece to complete the discovery of a whole OCBC model. All activities are discovered in the process of the discovery of AOC relationships. Therefore, the discovery of the activity model is to discover all possible behavioral constraints between activities.

From the modeling angle, behavioral constraints describe the possible orders between events. More precisely, a behavioral constraint specifies how many target events should occur before and after each reference event within a scope. The scope, i.e., the reference events and their target events, is identified by a correlation pattern. From the discovery angle, we can discover a behavioral constraint based on the observed events corresponding to a correlation pattern. The basic idea is shown in Figure 6.9. By mapping the input log onto a correlation pattern, the reference events and corresponding target events are identified. If the relations (in form of numbers) between the reference events and target events conform to the semantics of some constraint type, a constraint of this type is discovered.

According to the idea, a correlation pattern serves as the base to discover behavioral constraints. Therefore, before discovering behavioral constraints, we have to identify all possible correlation patterns. Once the correlation patterns are derived, the discovery of all constraints can be split into the discovery of constraints for each correlation pattern. Next, we define a function to extract all correlation patterns from the discovered class model and AOC relationships.

Definition 6.7 (Extracting Correlation Patterns) *Let* $ClaM = (C, R, \pi_1, \pi_2, \sharp_{src}^{\Box}, \sharp_{src}^{\Diamond}, \sharp_{tar}^{\Box}, \sharp_{tar}^{\Diamond})$ *be a class model and AOC be a set of AOC relationships. Function* $extP \in \mathcal{U}_{ClaM} \times \mathbb{P}(\mathcal{U}_{AOC}) \rightarrow \mathbb{P}(\mathcal{U}_P)$ *maps a class model and a set of AOC relationships onto a set of* correlation patterns *such that* $extP(ClaM, AOC) = VPatterns \cup UPatterns$ *where*

- $VPatterns = \{(a_{ref}, a_{tar}, c) \mid a_{ref} \neq a_{tar} \wedge \exists c \in C : \{(a_{ref}, c), (a_{tar}, c)\} \subseteq AOC\}$, *and*

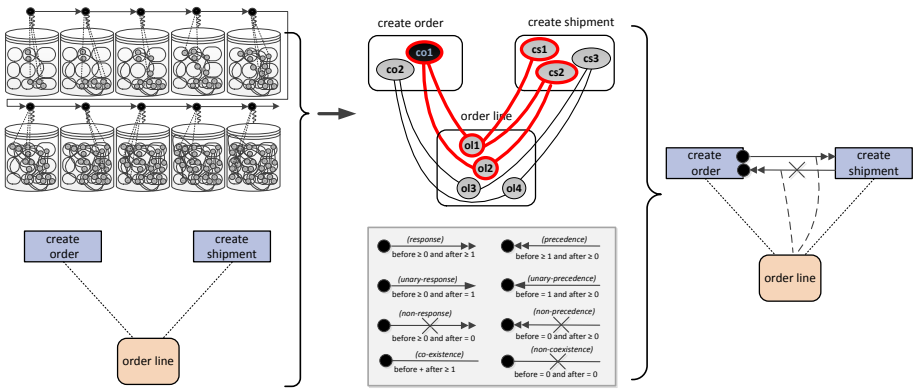


Figure 6.9: The idea of discovering behavioral constraints.

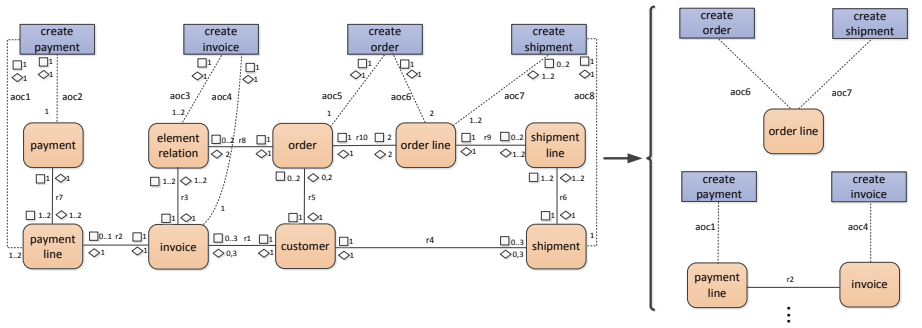


Figure 6.10: Extraction of event correlation patterns.

- $UPatterns = \{(a_{ref}, a_{tar}, r) \mid (\nexists c \in C : (a_{ref}, a_{tar}, c) \in VPatterns) \wedge a_{ref} \neq a_{tar} \wedge \exists r \in R : (\{(a_{ref}, \pi_1(r)), (a_{tar}, \pi_2(r))\} \subseteq AOC \vee \{(a_{ref}, \pi_2(r)), (a_{tar}, \pi_1(r))\} \subseteq AOC)\}$.

Correlation patterns consist of two types of patterns, i.e., V correlation patterns and U correlation patterns. A V correlation pattern (a_{ref}, a_{tar}, c) can be extracted if there exist two different activities a_{ref} and a_{tar}, referring to the same class c (i.e., $c \in C$). In contrast, a U correlation pattern (a_{ref}, a_{tar}, r) can be extracted if there exist two different activities a_{ref} and a_{tar}, referring to two classes connected by a class relationship r (i.e., $r \in R$). Consider for

example the class model *ClaM* and AOC relationships *AOC* in Figure 6.10 (on the left). A V correlation pattern $P5 = (create\ order, create\ shipment, order\ line)$ is extracted, since activities *create order* and *create shipment* refer to the same class *order line*, through two AOC relationships $aoc6$ and $aoc7$, respectively. Besides, a U correlation pattern $P1 = (create\ payment, create\ invoice, r2)$ is extracted, since (i) activity *create payment* refers to class *payment line* through $aoc1$, (ii) activity *create invoice* refers to class *invoice* through $aoc4$, and (iii) these two classes *payment line* and *invoice* are related through the class relationship $r2$. In this method, $extP(ClaM, AOC) = \{P1, P2, ..., P6\}$ as shown in Table 6.6.

Pattern	Reference activity	Target activity	Intermediary	
			Class	Class Relationship
P1	create payment	create invoice	-	r2
P2	create invoice	create payment	-	r2
P3	create invoice	create order	-	r8
P4	create order	create invoice	-	r8
P5	create order	create shipment	order line	-
P6	create shipment	create order	order line	-

Table 6.6: Extracted correlation patterns from the class model and AOC relationships in Figure 6.10.

Note that in this thesis we assume that a class has a priority over a class relationship to act as the intermediary for a correlation pattern. In other words, for a reference activity and a target activity, if there exist both a class and a class relationship which can be the intermediary, only the V pattern with the class is taken into consideration by function *extP*. For instance, for the reference activity *create order* and the target activity *create shipment*, there exists a V pattern $(create\ order, create\ shipment, order\ line)$ with the class *order line* as the intermediary. Therefore, the U pattern $(create\ order, create\ shipment, r10)$ is discarded to avoid redundant patterns (a redundant pattern refers to a U pattern, that has the same reference and target activities as an existing V pattern). Correlation patterns always come in pairs. A pair of correlation patterns have the same intermediary and reverse reference and target activities. For instance, $P5$ and $P6$ are a pair of patterns.

A correlation pattern specifies a reference activity and a target activity, and indicates the paths to correlate target events to each reference event. Given a correlation pattern, the corresponding objects, object relations and references

can be combined into a path to correlate events. After event correlation, each reference event and its related target events form an instance corresponding to the correlation pattern. More precisely, for a reference event, the set of its target events E_{tar} is derived as follows:

- if the correlation pattern $P = (a_{ref}, a_{tar}, cr)$ is a V pattern, i.e., the correlation "bridge" cr is a class, target events for a reference event are those a_{tar} events (i.e., $\partial_{a_{tar}}(E)$) which refer to the same object of class cr as the reference event does, i.e., there exist some object which is referred to by both the reference and target event (i.e., $o \in relate(e_{ref}) \cap relate(e_{tar})$), or
- if the correlation pattern $P = (a_{ref}, a_{tar}, cr)$ is a U pattern, i.e., the correlation "bridge" cr is a class relationship, target events for a reference event are those a_{tar} events which refer to an object which is related to another object (through an object relation of cr), referred to by the reference event, i.e., $\exists o_1 \in relate(e_{ref}), o_2 \in relate(e_{tar}) : \{(cr, o_1, o_2), (cr, o_2, o_1)\} \cap Rel_e \neq \emptyset)\}$.

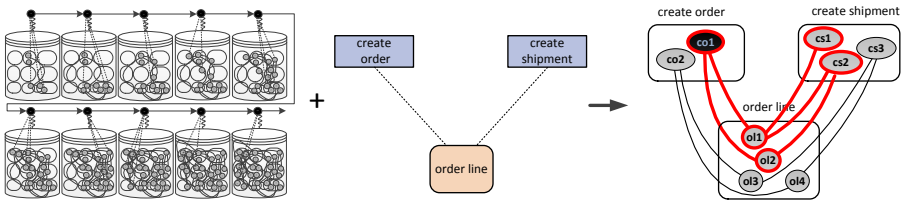


Figure 6.11: The reference event and target events are related through common objects.

Definition 6.8 (Event Correlation and Instances) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and $P = (a_{ref}, a_{tar}, cr)$ be a correlation pattern. Function $extI \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathbb{P}(E^*)$ correlates events in L based on P and returns a set of instances (i.e., event sequences), such that $extI(L, P) = \{ins \in E^* \mid (\exists e_{ref} \in \partial_{a_{ref}}(E) : \partial_{set}(ins) = \{e_{ref}\} \cup E_{tar}) \wedge (\forall 1 \leqslant i < j \leqslant |ins| : ins_i \prec ins_j)\}$ where* [4]

- *$E_{tar} = \{e_{tar} \in \partial_{a_{tar}}(E) \mid \exists e \in E, o \in \partial_{cr}(Obj_e) : o \in relate(e_{ref}) \cap relate(e_{tar})\}$ if P is a V pattern, or*

[4]For a sequence σ, e.g., ins, σ_i refers to the i-th element of the sequence, $|\sigma|$ denotes the length of the sequence and $\partial_{set}(\sigma)$ converts the sequence into a set.

- $E_{tar} = \{e_{tar} \in \partial_{a_{tar}}(E) \mid \exists o_1 \in relate(e_{ref}), o_2 \in relate(e_{tar}), e \in E : \{(cr, o_1, o_2), (cr, o_2, o_1)\} \cap Rel_e \neq \emptyset\}$ *if* P *is a U pattern.*

For simplicity, we define $ins_{\downarrow P} = (before, after) = (|\lhd_{e_{ref}}(E_{tar})|, |\rhd_{e_{ref}}(E_{tar})|)$.

Next, we explain the two rules in Definition 6.8 based on the example log in Table 6.1. Given a correlation pattern $P5$ = (*create order, create shipment, order line*) in Table 6.6, we extract (from the example log) two reference events $co1$ and $co2$, and three target events $cs1$, $cs2$ and $cs3$ as shown in Figure 6.11. Consider the reference event $co1$ to understand how to derive its target events. Since P is a V pattern and the correlation bridge is *order line*, we only observe the references between objects of *order line* and the reference event or target events. As shown in the "Reference" column in Table 6.1, $co1$ refers to objects $ol1$ and $ol2$, $cs1$ refers to object $ol1$, $cs2$ refers to objects $ol1$ and $ol2$, and $cs3$ refers to objects $ol3$ and $ol4$. Since $cs1$ and $co1$ refer to the common object $ol1$, $cs1$ is one target event of $co1$. Similarly, $cs2$ is also one target event of $co1$.

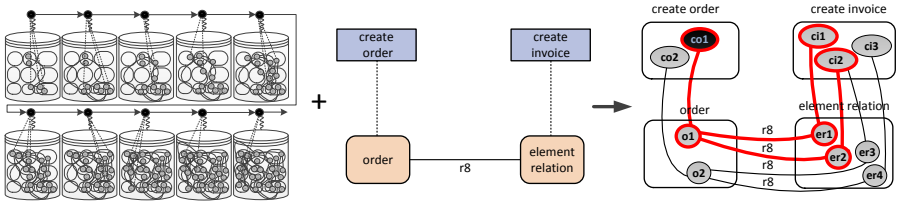


Figure 6.12: The reference event and target events are related through object relations.

Figure 6.12 shows how to correlate events based on a U pattern. Given a U pattern $P4$ = (*create order, create invoice, r*8) in Table 6.6, there are two reference events $co1$ and $co2$, and three target events $ci1$, $ci2$ and $ci3$. The explanation of correlation is still based on the reference event $co1$. Since P is a U pattern and the correlation bridge is the class relationship $r8$, the correlation paths consist of the references of AOC relationships (*create order, order*) and (*create invoice, element relation*), and the object relations of $r8$. As shown in the "Reference" column in Table 6.1, $co1$ refers to object $o1$, $ci1$ refers to object $er1$, $ci2$ refers to objects $er2$ and $er3$, and $ci3$ refers to object $er4$. Besides, $er1$ and $er2$ are connected to $o1$ through object relations of $r8$ as shown in the "Relations" column. Based on the above references and object relations, $ci1$ and $ci2$ are the target events of $co1$.

pattern	instance			shared
	instance1	instance2	instance3	events
P1	$\langle ci1, ci2, cp1\rangle$	$\langle ci3, cp2\rangle$	-	-
P2	$\langle ci1, cp1\rangle$	$\langle ci2, cp1\rangle$	$\langle ci3, cp2\rangle$	$cp1$
P3	$\langle co1, ci1\rangle$	$\langle co1, co2, ci2\rangle$	$\langle co2, ci3\rangle$	$co1$, $co2$
P4	$\langle co1, ci1, ci2\rangle$	$\langle co2, ci2, ci3\rangle$	-	$ci2$
P5	$\langle co1, cs1, cs2\rangle$	$\langle co2, cs3\rangle$	-	-
P6	$\langle co1, cs1\rangle$	$\langle co1, cs2\rangle$	$\langle co2, cs3\rangle$	$co1$

Table 6.7: Instances for all patterns in Table 6.6.

After event correlation, related events are integrated into instances. An instance *ins* is an event sequence ($ins \in E^*$). More precisely, all events in the instance ($\partial_{set}(ins)$) include a reference event (e_{ref}) and a set of target events (E_{tar}). Note that there are no duplicated events in one instance, i.e., any reference or target event cannot appear more than once in one instance. Each reference event corresponds to precisely one instance (i.e., reference events are "unique" and one reference event cannot appear in multiple instances). In contrast, target events may be split and replicated over multiple instances when they are correlated to multiple reference events. For instance, the event $ci2$ is correlated to both $co1$ and $co2$ in Figure 6.12. Therefore, $ci2$ is replicated over the resulting two instances corresponding to $P4$ in Table 6.7.

Besides, the order of events in the instance is consistent with the order of events in the log. Therefore, for any two events in the instance, ins_i (the i-th event of the instance) and ins_j (the j-th event of the instance), $i < j$ (ins_i is before ins_j in the instance), if and only if $ins_i \prec ins_j$ (ins_i is before ins_j in the log). Table 6.7 presents the instances corresponding to each correlation pattern in Table 6.6. For instance, for the correlation pattern $P5$, there are two instances $\langle co1, cs1, cs2\rangle$ and $\langle co2, cs3\rangle$, which are derived through the process in Figure 6.11.

By relating events based on a correlation pattern, we get a set of instances, which can be used to discover constraints corresponding to the pattern. The semantics of a constraint is defined as a restriction on the relations between events in a scope. On the other hand, from the discovery angle, a constraint can be identified as a tuple of a pattern and a constraint type, i.e., (P, ct). The pattern P specifies the scope, i.e., the reference events and target events, and the constraint type ct indicates the restriction on the relations between reference

events and target events.

Definition 6.9 (Specification of Behavioral Constraints) *Let $P = (a_{ref}, a_{tar}, cr)$ be a correlation pattern and ct be a constraint type. A behavioral constraint is specified as a tuple $(P, ct) \in \mathcal{U}_P \times \mathcal{U}_{CT}$. Function $idCon \in \mathcal{U}_P \times \mathcal{U}_{CT} \rightarrow \mathcal{U}_{Con}$ gives a constraint id referring to a constraint specification. Furthermore, function $speCon \in \mathcal{U}_{Con} \rightarrow \mathcal{U}_P \times \mathcal{U}_{CT}$ gives the constraint specification corresponding to a constraint id.*

$\mathcal{U}_{Con}$ (cf. Chapter 5) defines the universe of possible constraint ids, which does not specify the constraints. In our discovery approach, a constraint is specified as a combination of a correlation pattern and a constraint type (P, ct), which means that the events correlated by the correlation pattern P should follow the restriction indicated by the constraint type ct. We define two functions *idCon* and *speCon* to convert between constraint ids and specifications. For instance, if a constraint id $con1$ corresponds to a constraint specification (P, ct), $idCon((P, ct)) = con1$ and $speCon(con1) = (P, ct)$. In this thesis, a constraint id is called a constraint for simplicity, i.e., we omit the word "id" when there is no need to distinguish the id from the specification.

Indicated by the specification of a constraint (P, ct), a correlation pattern P may correspond to multiple constraints of different types. As shown in Figure 6.13, instances corresponding to a pattern P are checked with a given set of constraint types. If the relations between the reference events and target events conform to the semantics of some constraint type, a constraint of this type is discovered.

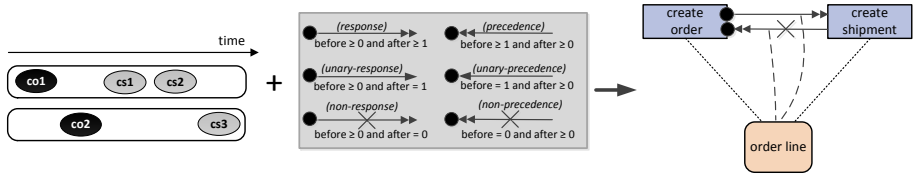


Figure 6.13: The idea of discovering behavioral constraints.

Definition 6.10 (Discovery of Behavioral Constraints) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log, $P = (a_{ref}, a_{tar}, cr)$ be a correlation pattern and CT be a set of constraint types. Function $disCon \in \mathcal{U}_L \times \mathcal{U}_P \times \mathbb{P}(CT) \rightarrow \mathbb{P}(\mathcal{U}_{Con})$*

discovers a set of behavioral constraints, such that $disCon(L, P, CT) = \{con \mid con = idCon(P, ct) \wedge ct \in CT \wedge \forall ins \in extI(L, P) : ins_{\restriction P} \in ct\}$

Given a log L, a correlation pattern P and a set of constraint types CT, function *disCon* discovers all constraints of CT corresponding to P from L. For instance, Figure 6.13 shows how to discover the constraints based on the pattern $P5 = (create\ order, create\ shipment, order\ line)$ and a set of six constraint types from the log in Table 6.1. After event correlation, we get two instances $\{\langle co1, cs1, cs2\rangle, \langle co2, cs3\rangle\}$. By checking all constraint types (i.e., the six types shown in the middle of Figure 6.13), these two instances conform to the semantics of *response* and *non-precedence* types. Therefore, two constraints are discovered: one of *response* and one of *non-precedence* type. By using this method, Table 6.8 presents all discovered constraints (indicated by "Yes") for each pattern in Table 6.6 from the log in Table 6.1. For instance, $(P1, precedence)$ (corresponding to "Yes") is a discovered constraint while $(P1, response)$ (corresponding to "No") is not.

pattern	constraint type					
	response	unary-response	non-response	precedence	unary-precedence	non-precedence
P1	No	No	Yes	Yes	No	No
P2	Yes	Yes	No	No	No	Yes
P3	No	No	Yes	Yes	No	No
P4	Yes	No	No	No	No	Yes
P5	Yes	No	No	No	No	Yes
P6	No	No	Yes	Yes	Yes	No

Table 6.8: Behavioral constraints discovered from the log in Table 6.1.

Definition 6.11 (Discovery of Activity Model) *Let L be a sound XOC event log, CT be a set of constraint types, $ClaM$ be a discovered class model and AOC be a discovered set of AOC relationships. An activity model discovered from L is a tuple $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$, where*

- $A \subseteq \mathcal{U}_A$ *is the set of activities such that* $A = \{act(e) \mid e \in E\}$,
- $Con \subseteq \mathcal{U}_{Con}$ *is a set of constraints such that* $Con = \{con \mid con \in disCon(L, P, CT) \wedge P \in extP(ClaM, AOC)\}$,
- $\pi_{ref} \in Con \rightarrow A$ *defines the reference activity of a constraint such that* $\forall con \in Con : \pi_{ref}(con) = a_{ref}$, *where* $((a_{ref}, a_{tar}, cr), ct) = speCon(con)$,

- $\pi_{tar} \in Con \rightarrow A$ *defines the target activity of a constraint such that* $\forall con \in Con : \pi_{tar}(con) = a_{tar}$*, where* $((a_{ref}, a_{tar}, cr), ct) = speCon(con)$*, and*
- $type \in Con \rightarrow \mathcal{U}_{CT}$ *defines the type of a constraint such that* $\forall con \in Con :$ $type(con) = ct$*, where* $((a_{ref}, a_{tar}, cr), ct) = speCon(con)$.

Definition 6.11 shows how to discover an activity model based on the discovered class model and AOC relationships. An activity is discovered if there is a corresponding event in the input log. Constraints are derived by (i) identifying all possible correlation patterns using function *extP* and (ii) discovering constraints corresponding to each pattern using function *disCon*. The top part of Figure 6.14 shows a discovered activity model with four activities and eight behavioral constraints (note that some of them are combined) from the log in Table 6.1.

6.1.5 Integrating Discovered Parts into an OCBC Model

In the previous sections, we have explained how to discover all the elements of an OCBC model. More precisely, Section 6.1.2 focused on discovering the data perspective, i.e., structure of objects and cardinality constraints while Section 6.1.3 discovered AOC relationships to build a bridge for discovering the behavioral perspective, i.e., behavioral constraints. Section 6.1.4 introduced how to relate events and discover behavioral constraints based on correlation patterns. Here, we give the formal definition of discovering an OCBC model by integrating all discovered parts.

Definition 6.12 (Discovery of OCBC Models) *Let* L *be a sound XOC event log and* $CT \subseteq \mathcal{U}_{CT}$ *be a set of constraint types. An OCBC model discovered from* L *is a tuple* $OCBCM = (ClaM, ActM, AOC, \natural_A^{\Box}, \natural_A^{\Diamond}, \natural_{OC}, crel)$*, where*

- *ClaM is the class model discovered from* L *(Definition 6.4),*
- *AOC is the set of AOC relationships, and* $\natural_A^{\Box}$*,* $\natural_A^{\Diamond}$ *and* $\natural_{OC}$ *are three cardinality functions discovered from* L *(Definition 6.6),*
- *ActM is the activity model discovered from* L *(Definition 6.11), and*
- $crel \in Con \rightarrow C \cup R$ *indicates the event correlation pattern of each constraint such that* $\forall con \in Con : crel(con) = cr$*, where* $((a_{ref}, a_{tar}, cr), ct) = speCon(con)$.

Figure 6.14 shows a discovered OCBC model from the example log in Table 6.1, which describes the data perspective, behavioral perspective and the interplay between them in a single diagram. It clearly reveals the involved classes, activities and constraints in the OTC scenario from which the example

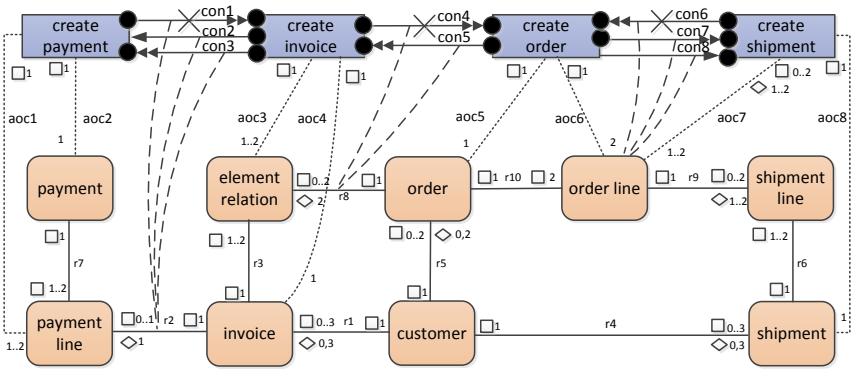


Figure 6.14: The discovered OCBC model from the motivating example, omitting the "eventually" cardinalities which are indicated by their corresponding "always" cardinalities.

log is generated. More precisely, the nine classes and ten class relationships make up the backbone of the OCBC model, i.e., the data perspective. They indicate the types of involved objects and restrictions which are followed by objects in the real transactions. The activity model contains eight (some of them combined) behavioral constraints (i.e., $con1$,...,$con8$) which specify the temporal restrictions on behavioral perspective. Between the class model and the activity model, eight AOC relationships (i.e., $aoc1$,...,$aoc8$) relate these two perspectives.

Note that the input log for discovery approach may only cover a segment of the whole data involved in a business process. In other words, the discovered model may be overfitting, especially in terms of the cardinality constraints. For instance, in the real business process, one order can have multiple order lines. In the example log, we only observe two orders, each of them corresponding to precisely two order lines. Therefore, the discovered cardinality is "2", which can not be true if we observe more data. In order to generalize the discovered model, we can expand the discovered cardinalities. Note that the expansion increases the generalization, but decreases the precision of the model. An extreme solution is to expand all cardinalities as "*", which results in a general but underfitting model. In order to make a balance, we import some normalized cardinalities in data modeling languages and generalize the discovered cardinalities as normalized ones.

Definition 6.13 (Normalization of Cardinalities) *Let* $\{0\}$, $\{1\}$, $\{0,1\}$, $\{0,1,...\}$ *and*

$\{1,2,...\}$ *(denoted as* 0*,* 1*,* 0..1*,* * *and* 1..**, respectively in the graph) be five normalized cardinalities. Any discovered cardinality card can be normalized as one of these cardinalities as follows:*

- $\{0\}$ *if* $card = \{0\}$*, or*
- $\{1\}$ *if* $card = \{1\}$*, or*
- $\{0,1\}$ *if* $card = \{0,1\}$*, or*
- $\{0,1,...\}$ *if* $0 \in card \wedge \exists n \in card : n > 1$*, or*
- $\{1,2,...\}$ *if* $0 \notin card \wedge \exists n \in card : n > 1$.

Table 6.9 shows some examples of normalization of cardinalities. For instance, the cardinality "1..2" on the relation $r7$ is normalized as "1..*", since the cardinality contains a number larger than one (i.e., 2) and do not contain zero.

relation	cardinality	notation in model	normalized cardinality	normalized notation
aoc2	{1}	1	{1}	1
r7	{1,2}	1..2	{1,2,...}	1..*
r2	{0,1}	0..1	{0,1}	0..1
r8	{0,1,2}	0..2	{0,1,...}	*
r8	{2}	2	{1,2,...}	1..*
r1	{0,1,2,3}	0..3	{0,1,...}	*
r1	{0,3}	0,3	{0,1,...}	*

Table 6.9: Some examples of normalization of cardinalities.

Note that in this section, we focus on discovering models from "clean" logs without any noise. However, the basic discovery algorithm can be easily extended to deal with noise by setting thresholds. For instance, in the discovery of AOC relationship in Definition 6.6, the basic algorithm discovers an AOC relationship (a, c) between the activity a and the class c, if there exists "at least one" a event which refers to a c object. It is possible to set a threshold on the number of a events referring to c objects. In other words, we can replace "at least one" with a number (e.g., 10, indicating 10 a events) or a ratio (e.g., 20%, indicating 20% of all events).

6.2 Advanced Discovery Techniques to Deal with Noise

Model discovery techniques take an event log as input and return a model to reveal the real business process where the log is generated. The quality of the input log decides if the discovered model can really represent the real business

process. In other words, to discover a representative model, the event log should contain a representative sample of data. A sample may be not suitable due to problems from two perspectives: (i) it has "too little data" to cover various events and objects in the process; (ii) it has "too much data" with events or objects unrelated to the target process. In process mining, the first perspective refers to the "incompleteness" problem while the second perspective refers to the "noise" problem.

The "incompleteness" is touched a little in Section 6.1.5, where we generalize cardinalities by expanding discovered cardinalities to normalized ones. In this section, we focus on dealing with the "noise" problem, which is significant in process mining and should be faced by any applicable discovery algorithm. Note that the noise here refers to the infrequent events or objects rather than the incorrect logging, since we assume all the data in logs are correctly recorded.

The approach in Section 6.1 cannot deal with noise and it mainly suffers two problems when facing noise. First, the noise may contain a variety of infrequent events or objects unrelated to the target business process, which makes the discovered models too complex (i.e., too many entities and constraints). Besides, the discovered constraints are not precise, since the approach forces the constraints to allow the occurrence of noise. In order to solve these problems, we next introduce some more advanced approaches to deal with noise in logs by (i) simplifying complex discovered models, (ii) discovering precise behavioral constraints and (ii) discovering precise cardinality constraints.

6.2.1 Simplification of OCBC Models

The approach in Section 6.1 is sensitive to noise, e.g., one object of an infrequent class leads to the discovery of the "noise" class. As a result, the noise makes the useful insights hiding in the discovered complex model. In this section, we give solutions to simplify OCBC models, i.e., filtering the unnecessary entities and constraints corresponding to the infrequent events or objects, to make the insights outstanding.

Definition 6.14 (Support) *Let* $L = (E, act, attrE, relate, om, \preceq)$ *be a sound XOC event log,* $M = (ClaM, ActM, AOC, \sharp_A^{\square}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ *be the discovered OCBC model where* $ClaM = (C, R, \pi_1, \pi_2, \sharp_{src}^{\square}, \sharp_{src}^{\Diamond}, \sharp_{tar}^{\square}, \sharp_{tar}^{\Diamond})$ *is the class model and* $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ *is the activity model.* $Ele_M = C \cup R \cup A \cup AOC \cup Con$ *is the set of all elements in the model. Function* $support \in Ele_M \rightarrow [0,1]$ *maps an element* $ele \in Ele_M$ *to a value between 0 and 1. The function has five versions to calculate the support of different types of elements.*

The support of a class is defined as the fraction of objects of the class in all objects, i.e., for each $c \in C$,

$$supportC(c) = \frac{|\{o \mid \exists e \in E : (o \in Obj_e \wedge class_e(o) = c)\}|}{|\{o \mid \exists e \in E : o \in Obj_e\}|}.$$

The support of a class relationship is defined as the fraction of object relations corresponding to the relationship in all object relations, i.e., for each $r \in R$,

$$supportR(r) = \frac{|\{(r, o_1, o_2) \mid \exists e \in E : (r, o_1, o_2) \in Rel_e\}|}{|\{rel \mid \exists e \in E : rel \in Rel_e\}|}.$$

The support of an activity is defined as the fraction of events of the activity in all events, i.e., for each $a \in A$,

$$supportA(a) = \frac{|\{e \in E \mid act(e) = a\}|}{|E|}.$$

The support of an AOC relationship is defined as the fraction of references corresponding to the relationship in all references, i.e., for each $aoc = (a, c) \in AOC$,

$$supportAOC(aoc) = \frac{|\{(e, o) \mid e \in E \wedge o \in relate(e) \wedge act(e) = a \wedge class_e(o) = c\}|}{|\{(e, o) \mid e \in E \wedge o \in relate(e)\}|}.$$

The support of a behavioral constraint is defined as the fraction of instances (of P) which consist with the constraint, in instances of all patterns, i.e., for each $con \in Con$,

$$supportCon(con) = \frac{|\{ins \in extI(L, P) \mid ins_{\downarrow P} \in type(con)\}|}{\sum_{P' \in extP(ClaM, AOC)} |extI(L, P')|}$$

where $P = (\pi_{ref}(con), \pi_{tar}(con), crel(con))$ *is the corresponding pattern of con.*

Consider the log in Table 6.1 and the model in Figure 6.14 to understand how to compute the support of different elements. For instance, $supportC(order) = 2/28$ since there are two *order* objects and twenty-eight objects in total. $supportR(r10) = 4/36$ since there are four object relations of $r10$ and thirty-six object relations in total. $supportA(create\ order) = 2/10$ since there are two *create order* events and ten events in total. $supportAOC(aoc5) = 2/26$ since there are two references of $aoc5$ (between *create order* events and *order* objects) and twenty-six references in total. $supportCon(con2) = 3/15$ since there are three consistent instances of $P2$ (corresponding pattern of *con2*) and fifteen instances in total as shown in Table 6.7.

Note that the support in Definition 6.14 is a relative support, i.e., a fraction between 0 and 1. It is possible to compute an absolute support by only considering the numerator of each formula, e.g., $supportC(c) = |\{o \mid \exists e \in E : (o \in Obj_e \wedge class_e(o) = c)\}|$ for a class c. Table 6.10 presents the support of all classes in Figure 6.14, i.e., the "Instance number" column shows the absolute support and the "Support" column shows the relative support. One can specify a threshold for support to filter out any element whose support is below the threshold. In this way, we can simplify the complex OCBC model discovered in the environment with noise.

Class	Instance number	Support	R_{del}	AOC_{del}	Con_{del}
payment	2	2/28	r7	aoc2	-
payment line	3	3/28	r7,r2	aoc1	con1,con2,con3
element relation	4	4/28	r3,r8	aoc3	con4,con5
invoice	3	3/28	r1,r3	aoc4	con1,con2,con3
order	2	2/28	r8,r10	aoc5	con4,con5
customer	2	2/28	r1,r4,r5	-	-
order line	4	4/28	r9,r10	aoc6,aoc7	con6,con7,con8
shipment line	5	5/28	r6,r9	-	-
shipment	3	3/28	r4,r6	aoc8	-

Table 6.10: Support and related elements of classes.

Definition 6.15 (Filtering Infrequent Elements in OCBC Model) *Let L be a sound XOC event log, M be the discovered OCBC model and $Ele_M = C \cup R \cup A \cup AOC \cup Con$ be the set of all elements in the model. Let τ be a threshold. An element in the model $ele \in Ele_M$ is filtered out, if its corresponding support is below the threshold, i.e., $support(ele) < \tau$ and all its related elements are removed too as follows:*

- *if $ele = c_{del} \in C$, the related elements are $R_{del} = \{r \in R \mid \pi_1(r) = c \vee \pi_2(r) = c\}$, $AOC_{del} = \{(a, c_{del}) \in AOC\}$ and $Con_{del} = \{con \in Con \mid crel(con) = c_{del} \vee crel(con) \in R_{del}\}$,*
- *if $ele = r_{del} \in R$, the related elements are $Con_{del} = \{con \in Con \mid crel(con) = r_{del}\}$,*
- *if $ele = a_{del} \in A$, the related elements are $AOC_{del} = \{(a_{del}, c) \in AOC\}$ and $Con_{del} = \{con \in Con \mid \pi_{ref}(con) = a_{del} \vee \pi_{tar}(con) = a_{del}\}$,*
- *if $ele = (a, c)_{del} \in AOC$, the related elements are $Con_{del} = \{con \in Con \mid (\pi_{ref}(con) = a \vee \pi_{tar}(con) = a) \wedge (crel(con) = c \vee \pi_1(crel(con)) = c \vee \pi_2(crel(con)) = c)\}$, and*
- *if $ele = con_{del} \in Con$, there are no related elements.*

Note that in the process of discovering an OCBC model, the elements are discovered in the order: C, R, A, AOC and Con. As a result, the latter discovery

may depend on the former discovery result. For instance, the discovery of class relationships depends on the discovered classes. Therefore, when an element *ele* is filtered out, all its related elements (i.e., elements discovered based on *ele*) should be removed too. Based on the type of *ele*, Definition 6.15 gives five rules to specify the related elements.

If *ele* is a class c_{del}, the related elements consist of (i) all class relationships connected to the class (R_{del}), (ii) all AOC relationships connected to the class (AOC_{del}), and (iii) all behavioral constraints which correlate events by c_{del} or a class relationship which will be removed (Con_{del}). If *ele* is a class relationship r_{del}, the related elements only consist of behavioral constraints which correlate events by r_{del}. If *ele* is an activity a_{del}, the related elements contain AOC relationships which are related to the activity ($AOC_{del} = \{(a_{del}, c) \in AOC\}$) and behavioral constraints which have the activity as reference or target activity ($\pi_{ref}(con) = a_{del} \vee \pi_{tar}(con) = a_{del}$). If *ele* is an AOC relationship $(a, c)_{del}$, any behavioral constraint *con* which is connected to a (i.e., $\pi_{ref}(con) = a \vee \pi_{tar}(con) = a$), and correlate events by c (i.e., $crel(con) = c$) or a class relationship connected to c (i.e., $\pi_1(crel(con)) = c \vee \pi_2(crel(con)) = c$) will be removed too. Since the constraints are the last elements to be discovered, removing a constraint will not influence anything else.

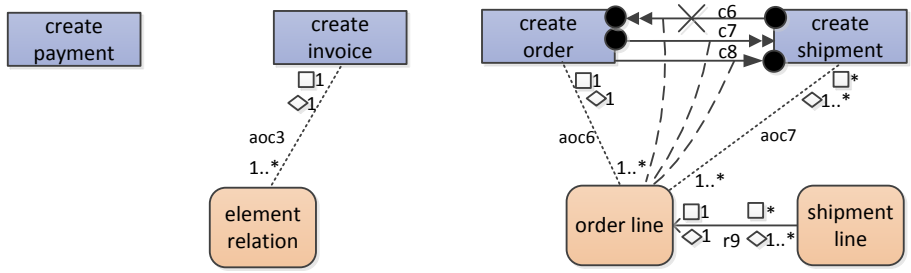


Figure 6.15: Filtering elements in an OCBC model based on support.

For instance, for the class *payment* in Figure 6.14, its related elements are $r7$ and $aoc2$ since they are connected to *payment*. It has no related behavioral constraints since it is not involved in any correlation pattern. Table 6.10 presents the related elements of all classes in Figure 6.14. Based on the support and related elements in Table 6.10, one can set a threshold to filter classes. For instance, given a threshold 3.5/28, the classes *payment*, *payment line*, *invoice*,

order, *customer* and *shipment*, and their related elements are removed, resulting in the model in Figure 6.15. Note that it is also possible to set a different threshold for each type of elements. The rules in Definition 6.15 still hold in these situations.

6.2.2 Discovery of Precise Behavioral Constraints

In Section 6.2.1, we proposed an approach to simplify the complex models, which deals with the noise problem on the simplicity perspective. Besides the simplicity perspective, the noise may force the constraints to be general enough to allow its occurrence, resulting in imprecise constraints. In this section, we illustrate how to filter noise and discover precise behavioral constraints, i.e., dealing with the noise problem on the imprecision perspective.

A constraint is defined in the context of a correlation pattern. It indicates the restriction on the reference events and target events in instances correlated by the pattern. In the discovery process (using the initial approach in Section 6.1), the observed instances decide the discovered constraints, i.e., if all instances (corresponding to the correlation pattern) satisfy the semantics of some constraint type, a constraint of this type is discovered. The requirement that *all instances satisfy the semantics* is too strict in an environment with noise, since an infrequent behavior, which is only a violation of a constraint type, may ruin the discovery of the corresponding constraint. Therefore, it is necessary to propose a robust discovery approach of behavioral constraints, which can still discover a constraint event if some noise violates the constraint.

The basic idea is to filter the noise according to a threshold, and discover constraints based on the behavior without noise as shown in Figure 6.16. In this way, the discovered constraints are precise, i.e., they do not allow the occurrence of noise. In this chapter, the noise in terms of behavioral constraints refers to infrequent instances. We define the types (i.e., variants) of instances and count the frequency of each type to identify the noise. If the frequency of a type is below the configured threshold, the instances of the type are considered as noise and filtered out. The remaining frequent types (highlighted in black) are taken as input to discover behavioral constraints. Next, we define a variant matrix to represent all possible types of instances and count the frequency of each type.

Definition 6.16 (Variant Matrix) *A variant matrix V_{CT} is a set of nine disjoint constraint types which incorporate all the possible relations between a reference event and its target events.* $V_{CT} = \{(0;0), (1;0), (0;1), (1;1), (2+;0), (0;2+), (2+;1), (1;2+),$

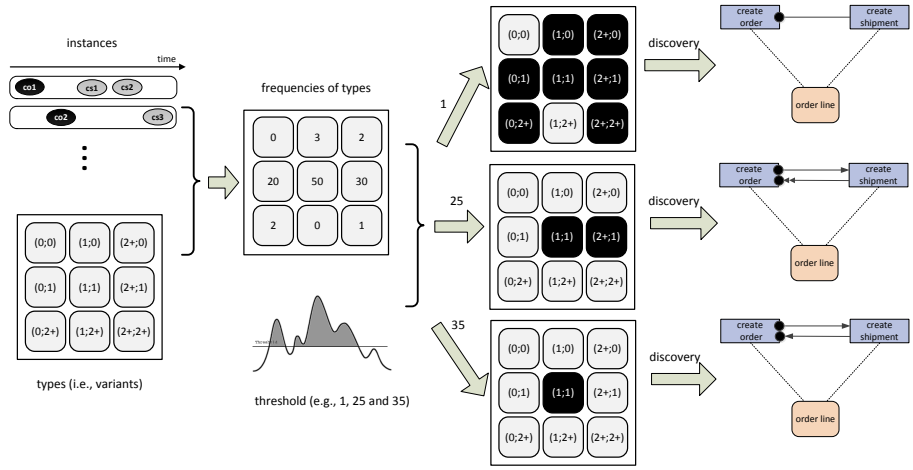


Figure 6.16: The idea to discover precise behavioral constraints from a log with noise.

$(2+;2+)\} \subseteq \mathcal{U}_{CT}$. *Each constraint type in the set is called a variant, e.g.,* $(1;1) \in V_{CT}$ *is a variant.*

The variant matrix contains nine different variants (cells), in the form of three rows and three columns as shown in Figure 6.17. Each variant in the matrix corresponds to a constraint type. For instance, $(1;0)$ is a constraint type which requires that the number of target events before (after) each reference event is one (zero). $(1;0)$ is the same as the *unary-precedence* constraint type (cf. Chapter 5). $(2+;0)$ is a constraint type which requires that the number of target events before each reference event is larger or equal to two, and there are no target events after each reference event. The formalization of these nine constraint types is shown in Table 6.11.

Note that the nine variants are disjoint, i.e., $\forall \nu, \nu' \in V_{CT} : \nu \neq \nu' \Rightarrow \nu \cap \nu' = \emptyset$. Besides, the nine variants cover all the possible relations between a reference event and its target events, i.e., each instance corresponds to precisely one variant, indicated by the relations between target events and the reference event in the instance. For example, if an instance has five target events before the reference event and precisely one target event after the reference event, i.e., $(5,1)$, it corresponds to variant $(2+;1)$ (since $(5,1) \in (2+;1)$). Note that $(5,1)$ is a pair of numbers (in which the former (latter) one represents the number of

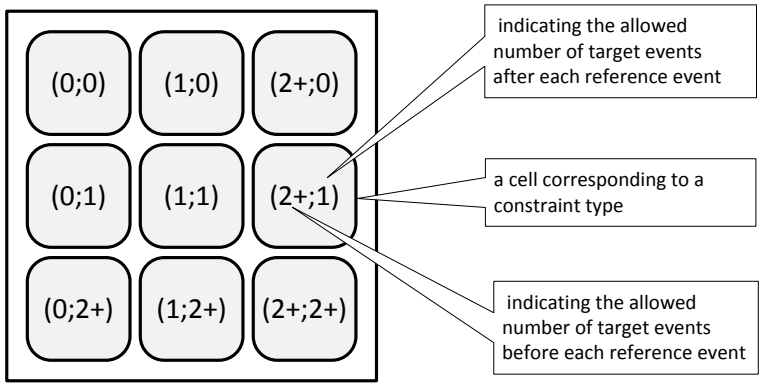


Figure 6.17: A variant matrix is represented by a grid consisting of 3x3 cells.

target events before (after) the reference event) while $(2+;1)$ is a set of pairs.

Definition 6.17 (Computing Variant Frequency) *Let L be a sound XOC log and P be a correlation pattern. Function $freV \in \mathcal{U}_L \times \mathcal{U}_P \times V_{CT} \rightarrow \mathbb{N}$ returns the frequency that a variant is observed in a log corresponding to a pattern such that $freV(L,P,v) = |\{ins \mid ins \in extI(L,P) \wedge ins_{\downarrow P} \in v\}|$.*

For convenience, we define the following shorthand. $freV_{\%}(L,P,v) = \frac{freV(L,P,v)}{\sum_{v \in V_{CT}} freV(L,P,v)}$ provides the ratio of a variant. For $V \subseteq V_{CT}$, $freV(L,P,V) = \sum_{v \in V} freV(L,P,v)$ and $freV_{\%}(L,P,V) = \sum_{v \in V} freV_{\%}(L,P,v)$ provides the frequency and ratio of multiple variants, respectively.

If an instance correlated by a correlation pattern P from a log L corresponds to a variant v, i.e., $ins_{\downarrow P} \in v$, we say that v is observed once in L. Based on this idea, Definition 6.17 defines a function to count how many times a variant is observed in a log corresponding to a pattern. More precisely, the frequency of a variant is equal to the number of instances which satisfy the requirements of the variant.

For instance, let L be the log example in Table 6.1 and P = (*create order*, *create shipment*, *order line*). After event correlation, we get two instances $\{\langle co1, cs1, cs2\rangle, \langle co2, cs3\rangle\}$ where $co1$ and $co2$ are reference events as shown in Figure 6.18. In the first instance, there are zero and two target events before and after the reference events, respectively, i.e., $\langle co1, cs1, cs2\rangle_{\downarrow P} = (0,2) \in (0;2+)$. Therefore $freV(L,P,(0;2+)) = 1$ and similarly $freV(L,P,(0;1)) = 1$ according to the

constraint variant	formalization
(0;0)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 0 \wedge after = 0\}$
(1;0)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1 \wedge after = 0\}$
(2+;0)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before > 1 \wedge after = 0\}$
(0;1)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 0 \wedge after = 1\}$
(1;1)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1 \wedge after = 1\}$
(2+;1)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before > 1 \wedge after = 1\}$
(0;2+)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 0 \wedge after > 1\}$
(1;2+)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1 \wedge after > 1\}$
(2+;2+)	$\{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before > 1 \wedge after > 1\}$

Table 6.11: The formalization of nine variants in the variant matrix (i.e., V_{CT}).

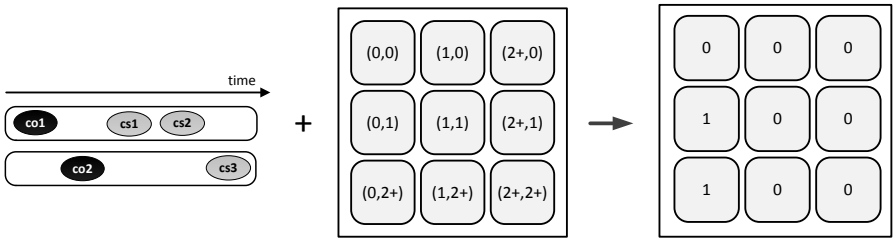


Figure 6.18: Computing the frequency of each variant based on instances, resulting in a *frequency matrix*.

second instance. Figure 6.18 shows the *frequency matrix*, i.e., frequencies of all variants observed in L corresponding to the pattern P. Based on the frequency matrix and a configured threshold, one can identify the frequent variants for discovery.

Definition 6.18 (Frequent Variants) *Let L be a sound XOC event log and P be a correlation pattern. $V_{fre} \subseteq V_{CT}$ is the frequent variants observed in L corresponding to P, such that for any $v \in V_{fre}$,*

- $freV(L, P, v) \geqslant \tau$ *for some threshold* $\tau \in \mathbb{N}$, *or*
- $freV_{\%}(L, P, v) \geqslant \tau$ *for some threshold* $\tau \in [0, 1]$, *or*
- $freV_{Entropy}(L, P, v) \geqslant \tau$ *for some threshold* $\tau \in [0, 1]$, *where* $freV_{Entropy}(L, P, v) = p \frac{\sum_{v' \in V_{CT}} -p' \log_2(p')}{\log_2(|V_{CT}|)}$, $p = freV_{\%}(L, P, v)$ *and* $p' = freV_{\%}(L, P, v')$.

Definition 6.18 provides three solutions to identify frequent variants by filtering infrequent variants (i.e., noise) according to a threshold. In the first solution, a variant is frequent if its frequency is equal to or larger than the given threshold (i.e., an integer). Figure 6.19(a) shows an example using the first solution to identify frequent variants. With a threshold 25, the variants $(1;1)$ and $(2+;1)$ are frequent (highlighted in black) since their frequencies (40 and 35) are above the threshold. The second solution employs a similar idea, only changing the absolute frequency to a relative ratio.

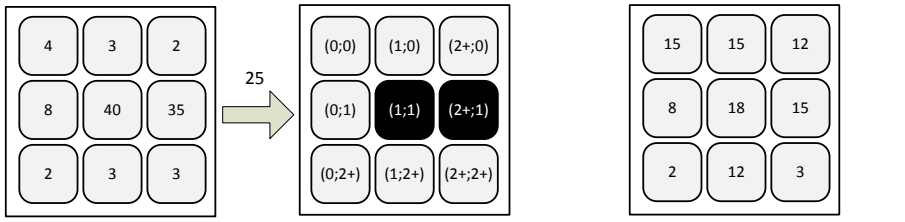


Figure 6.19: Identify frequent variants based on a threshold.

The first solution is based on the frequency of a individual variant and the second solution also considers the total frequency (i.e., a fraction of individual frequency in total frequency). They have limitations in some cases, where the frequencies of other variants should be considered when identifying if a variant is frequent. For instance, the variant $(0;1)$ has the same frequency (i.e., 8) in the frequency matrices in Figure 6.19(a) and Figure 6.19(b), and the frequency sums of all variants in these two matrices are also the same (i.e., 100). To identify if variant $(0;1)$ is frequent in these two different matrices, the first and second solutions return the same result given the same threshold. Apparently, $(0;1)$ is relatively more frequent in the second matrix, since the frequency differences between variants in the second matrix are smaller. In this situation, the entropy can be used to identify the frequent variants, as shown in the third solution.

Definition 6.19 (Discovery of Behavioral Constraints by Frequent Variants) *Let L be a sound XOC event log, P be a correlation pattern and CT be a set of constraint types. Function $disCN \in \mathcal{U}_L \times \mathcal{U}_P \times \mathbb{P}(CT) \rightarrow \mathbb{P}(\mathcal{U}_{Con})$ discovers behavioral constraints from a log with noise, such that $disCN(L, P, CT) = \{con \mid con = idCon(P, ct) \wedge ct \in CT_{fre} \setminus CT_{red}\}$ where*

- $CT_{fre} = \{ct \in CT \mid \forall v \in V_{fre} : v \subseteq ct\}$, *where* $V_{fre} \subseteq V_{CT}$ *is the frequent variants observed in the log, and*
- $CT_{red} = \{ct \in CT_{fre} \mid \exists ct' \in CT_{fre} : ct' \subsetneq ct\}$, *which is the set of redundant constraint types.*

If all frequent variants consist with a constraint type ct (i.e., $\forall v \in V_{fre} : v \subseteq ct$), a constraint of this type is discovered, i.e., $con = idCon(P, ct)$. Note that it is possible that there exist overlapping constraint types. For instance, the *response* type contains the *unary-response* type (cf. Chapter 5), i.e., the *response* type is looser. If two types are contained by CT_{fre} and one contains the other, the looser type is called a redundant constraint type. When discovering constraints, redundant types are discarded. For instance, there are three constraints of *unary-response*, *response* and *co-existence* types in Figure 6.20(a). After discarding the redundant constraints, only the strictest constraint, i.e., the *unary-response* constraint, remains as shown in Figure 6.20(b).

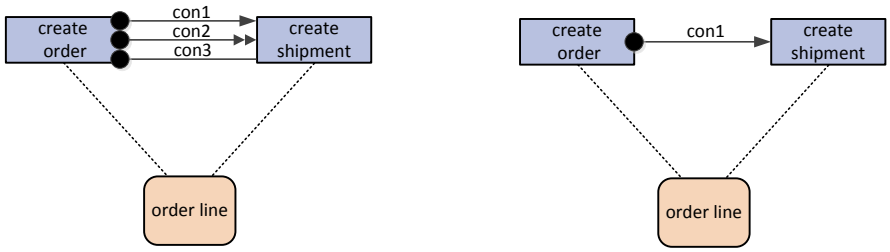


Figure 6.20: Discarding redundant constraints.

6.2.3 Discovery of Precise Cardinality Constraints

In Section 6.2.2, we illustrate how to discover precise behavioral constraints from a log with noise. In this section, a similar idea is used to discover precise cardinality constraints in an environment with noise.

In an OCBC model, cardinality constraints exist on class relationships and AOC relationships, as shown in Figure 6.21 (cf. Chapter 5). A cardinality constraint indicates the possible correspondence between two types of instances,

e.g., one-to-many. For instance, the cardinality constraint $card4$ on the *customer* side specifies a restriction on objects of *order* (the opposite side), i.e., each *order* object corresponds to precisely one *customer* object. Based on the semantics of cardinality constraints, each cardinality constraint has a reference and a target, and the constraint specifies the allowed numbers of target instances of each reference instance. Consider for example $card4$. The reference is *order* and the target is *customer*. Besides, cardinality constraints may be of different types, e.g., $card4$ is of "always" type and $card3$ is of "eventually" type.

The discussion above shows that there exist various cardinality constraints. Different cardinality constraints may represent different types of restrictions (e.g., "always" and "eventually") on different types of instances (e.g., classes and activities). Despite these differences, the contents of cardinality constraints (e.g., "0..1" and "1..*") are of the same essence, i.e., a set of integers. In this section, we focus on the discovery of the contents of various cardinality constraints. More precisely, we want to propose a generic solution to filter noise and discover the contents in a precise manner. Based on this idea, we next define a universe of cardinality constraints, including all types of cardinality constraints, and several functions to specify different perspectives.

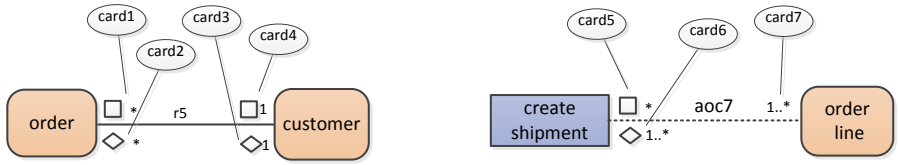


Figure 6.21: Different types of cardinality constraints on class relationships and AOC relationships.

Definition 6.20 (Cardinality Constraint Notations) *Let $\mathcal{U}_{CardId}$ be the universe of cardinality constraints (ids), $R \subseteq \mathcal{U}_R$ be a set of class relationships, $AOC \subseteq \mathcal{U}_{AOC}$ be a set of AOC relationships, $A \subseteq \mathcal{U}_A$ be a set of activities and $C \subseteq \mathcal{U}_C$ be a set of classes.*

- *$rCard \in \mathcal{U}_{CardId} \rightarrow R \cup AOC$ maps each cardinality constraint onto a class relationship or an AOC relationship where the cardinality constraint locates.*
- *$refCard \in \mathcal{U}_{CardId} \rightarrow A \cup C$ maps each cardinality constraint onto its reference, i.e., a class or an activity.*

- *tarCard* $\in \mathcal{U}_{CardId} \rightarrow A \cup C$ *maps each cardinality constraint onto its target, i.e., a class or an activity.*
- *typeCard* $\in \mathcal{U}_{CardId} \rightarrow \{\Box, \Diamond, \bigcirc\}$ *maps each cardinality constraint onto a type, where* $\Box$ *refers to an "always" cardinality constraint,* $\Diamond$ *refers to an "eventually" cardinality constraint and* $\bigcirc$ *refers to a cardinality constraint on the class side of an AOC relationship.*
- *conCard* $\in \mathcal{U}_{CardId} \rightarrow \mathcal{U}_{Card}$ *maps each cardinality constraint onto its contents, i.e., a set of integers.*

Consider the cardinality constraints in Figure 6.21 to understand the functions in Definition 6.20. For $card4$, *rCard*($card4$) = $r5$, *refCard*($card4$) = *order*, *tarCard*($card4$) = *customer*, *typeCard*($card4$) = $\Box$ and *conCard*($card4$) = $\{1\}$. For $card7$, *rCard*($card7$) = $aoc7$, *refCard*($card7$) = *create shipment*, *tarCard*($card7$) = *order line*, *typeCard*($card7$) = $\bigcirc$ and *conCard*($card7$) = $\{1,2,...\}$. Note that since there is only one cardinality constraint on the class side of an AOC relationship, the type of the constraint, i.e., the symbol $\bigcirc$, is omitted in the graph. *conCard* gives the content of a cardinality constraint. In this thesis, we use a cardinality to refer to a cardinality constraint (e.g., $card7$) or the content of a cardinality constraint (e.g., $1..^*$) when there is no need to distinguish them.

In our approach, the discovery of cardinality constraints comes after its corresponding class relationship or AOC relationship is discovered. In other words, the functions *rCard*, *refCard*, *tarCard*, and *typeCard* are known. The task is to discover the function *conCard*, i.e., the contents of cardinality constraints. For instance, assume a task is to discover the "eventually" cardinality constraint on the activity side of $aoc7$. For this task, we know *rCard* = $aoc7$, *refCard* = *order line*, *tarCard* = *create shipment*, and *typeCard* = $\Diamond$, and we need to discover *conCard*, i.e., the allowed numbers of *create shipment* events for each *order line* object at the last moment (i.e., eventually).

From the modeling angle, a cardinality constraint indicates the allowed numbers of target instances for each reference instance. From the discovery angle, we can integrate the observed numbers to discover *conCard*. More precisely, according to the type of the cardinality constraint, we use corresponding correlation function (e.g., *corR* or *corAOC* defined in Section 6.1) to correlate target instances to each reference instance. In the basic discovery approach, all numbers of target instances are put into the discovered cardinality constraint, no matter how frequent the numbers are. This method cannot deal with noise, i.e., infrequent numbers. For instance, if each reference instance has precisely one target instance except one reference instance has zero target instance, this method still discovers the number zero for the cardinality constraint.

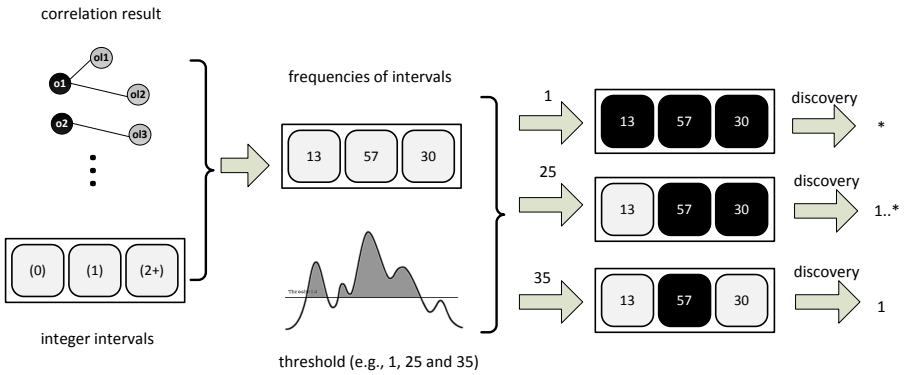


Figure 6.22: The idea to discover precise cardinality constraints from logs with noise.

In order to discover robust cardinality constraints, we take into consideration the frequencies of numbers. As shown in Figure 6.22, based on the correlation result, we compute the frequency of each number. Note that a reference instance may have any non-negative number of target instances such that the variety of numbers may be infinite in theory. For simplicity, we set some integer intervals to group numbers and compute the frequency of each interval. If the frequency of an interval is below the configured threshold, the numbers in this interval are considered as noise and filtered out. The remaining frequent intervals (highlighted in black) are taken as input to discover cardinality constraints. Next, we define an interval array (i.e., a set of intervals) to group numbers.

Definition 6.21 (Interval Array) *An interval array* $I_{Card} = \{(0),(1),(2+)\} \subseteq \mathcal{U}_{Card}$ *is a set of three disjoint cardinalities, where* $(0) = \{0\}$, $(1) = \{1\}$ *and* $(2+) = \{2,3,...\}$. *Each cardinality in the set is called an interval, e.g.,* $(2+) \in I_{Card}$ *is an interval.*

The interval array contains three cells, that correspond to three cardinalities, as shown in Figure 6.23(a). It incorporates all possible numbers of target instances of any reference instance, i.e., $(0) \cup (1) \cup (2+) = \mathbb{N}$. Besides, these three intervals are disjoint, i.e., $\forall i, i' \in I_{Card} : i \neq i' \Rightarrow i \cap i' = \emptyset$. Note that the interval array indicates the discovered cardinality constraints by grouping numbers. For instance, the interval array here, i.e., $\{(0),(1),(2+)\}$, indicates that we can discover the normalized cardinality constraints, which is explained later. Figure 6.23(b) shows an interval array filled with numbers, and each of them indicates the

number of observations of the corresponding interval in logs.

(a) interval array (b) frequencies of intervals

Figure 6.23: An interval array I_{Card} consists of three cardinalities and can be filled with numbers, which indicate the number of observations of that interval in logs.

Based on the idea in Figure 6.22, we map the correlation result onto the interval array to compute the frequencies of all intervals. If there exists a reference instance such that the number of its target instances is contained by some interval, we consider that the interval is observed once. It is easy to compute the frequencies when discovering $\diamondsuit$ and $\bigcirc$ cardinality constraints, since we only need to consider the number of the target instances of each reference instance at a point in time, i.e., the number is fixed. In contrast, it is difficult to compute the frequencies when discovering $\Box$ cardinality constraints, since we have to consider the numbers of target instances at all moments (after the reference instance is created), and the numbers may change over time.

For instance, considering that, for a certain time period, a reference instance starts with one target instance and, after a while, it has two target instances, we need to determine the number of target instances for this reference. In Definition 6.22, we give a solution to deal with this situation when discovering $\Box$ cardinality constraints.

Definition 6.22 (Identifying Normalized $\Box$ Interval Frequency) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and O_L be the set of all objects in L. Function $freIO \in \mathscr{U}_L \times \mathscr{U}_{CardId} \times I_{Card} \times O_L \rightarrow \mathbb{R}$ returns the frequency that an interval $i \in I_{Card}$ is observed in L, in terms of a reference object $o \in O_L$ corresponding to an "always" cardinality constraint $card \in \mathscr{U}_{CardId}$, such that*

$$freIO(L, card, i, o) = \frac{|\{e \mid \exists e \in E : o \in Obj_e \wedge |cor(o, r, e) \in i|\}|}{|\{e \mid \exists e \in E : o \in Obj_e\}|}.$$

where $r = rCard(card)$ and cor is (i) the function $corR$ if $r \in R$ or (ii) the function $corAOC$ if $r \in AOC$.

In the correlation result derived based on a cardinality constraint $card$ and a log L, a reference instance may have different numbers of target instances over time. Definition 6.22 gives a solution to compute the frequency that each interval is observed in terms of such a reference instance. More precisely, the frequency of i in terms of an object o is the ratio of moments, when the number of target instances is contained by i, to all moments when o exists. Since $\square$ cardinality constraints locate on a class relationship or an AOC relationship, we choose the corresponding function to identify the target instances, i.e., *corR* for a class relationship or *corAOC* for an AOC relationship.

Object	Number of target instances $\lvert cor(o,r,e_i)\rvert$										frequency *freIO*		
	$co1$	$co2$	$cs1$	$ci1$	$cs2$	$ci2$	$cs3$	$cp1$	$ci3$	$cp2$	(0)	(1)	(2+)
$o1$	1	1	1	1	1	1	1	1	1	1	0	1	0
$o2$	-	1	1	1	1	1	1	1	1	1	0	1	0
$c1$	1	2	2	2	2	2	2	2	2	2	0	1/10	9/10
$c2$	0	0	0	0	0	0	0	0	0	0	1	0	0

Table 6.12: The numbers of target instances of each *order* or *customer* object at each moment (represented by an event) and frequencies of each interval for an object.

Consider the "always" cardinality constraint $card1$ in Figure 6.21 and the example log L in Table 6.1 to understand how to compute the interval frequencies in terms of an object. The reference instances of $card1$ are objects $c1$ and $c2$. The numbers of target instances of $c1$ or $c2$ at all moments are shown in Table 6.12. L has ten moments corresponding to ten events. $c1$ has one target instance at the moment that $co1$ happens and two target instances at all other moments. We do not observe that $c1$ has no target instances at any moment. Therefore, $freIO(L, card1, (0), c1) = 0$, $freIO(L, card1, (1), c1) = 1/10$ and $freIO(L, card1, (2+), c1) = 9/10$.

Definition 6.23 (Computing Interval Frequency) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and card be a cardinality constraint. Function $freI \in \mathcal{U}_L \times \mathcal{U}_{CardId} \times I_{Card} \rightarrow \mathbb{R}$ returns the frequency that an interval $i \in I_{Card}$ is observed in a log $L \in \mathcal{U}_L$ corresponding to a cardinality constraint $card \in \mathcal{U}_{CardId}$, such that*

- *if card is an "always" cardinality constraint, i.e., $typeCard(card) = \square$,*

$$freI(L, card, i) = \sum_{o \in \partial_c(O_L)} freIO(L, card, i, o)$$

where $c = refCard(card)$ is the reference, or

- *if card is an "eventually" cardinality constraint, i.e., $typeCard(card) = \diamondsuit$,*

$$freI(L, card, i) = |\{o \in \partial_c(O_L) \mid |cor(o, r, e_l)| \in i\}|$$

where $c = refCard(card)$ is the reference, $r = rCard(card)$ is the relationship, e_l is the last event in L and cor is (i) the function corR if $r \in R$ or (ii) the function corAOC if $r \in AOC$, or

- *if card is a $\bigcirc$ cardinality constraint, i.e., $typeCard(card) = \bigcirc$,*

$$freI(L, card, i) = |\{e \in \partial_a(E) \mid |\{o \in \partial_c(relate(e))\}| \in i\}|$$

where $a = refCard(card)$ is the reference and $c = tarCard(card)$ is the target.

In order to discover a cardinality constraint $card$ from a log L, we map L onto the interval array and compute the frequency of each interval based on Definition 6.23. More precisely, for an "always" cardinality constraint, the frequency of an interval i is the sum of a set of values, in which each one is the frequency that i is observed for a reference instance. For instance, $freI(L, card1, (0)) = 1$, since $freIO(L, card1, (0), c1) = 0$ and $freIO(L, card1, (0), c2) = 1$ based on the values in Table 6.12. Similarly, $freI(L, card1, (1)) = 1/10$ and $freI(L, card1, (2)) = 9/10$.

For an "eventually" cardinality constraint, the reference is a class c. The frequency of an interval i is the number of c objects, in which one object has n (i.e., $|cor(o, r, e_l)|$) target instances at the last moment (i.e., e_l) and $n \in i$. Note that since an "eventually" cardinality constraint locates on a class relationship or an AOC relationship, we choose a corresponding function to identify the target instances, i.e., *corR* for a class relationship or *corAOC* for an AOC relationship.

If a cardinality constraint is of type $\bigcirc$, the reference is an activity a and the target is a class c. The frequency of an interval i is the number of a events (i.e., $e \in \partial_a(E)$), in which one event refers to n (i.e., $|\{o \in \partial_c(relate(e))\}|$) objects of c and $n \in i$.

Definition 6.24 (Frequent Intervals) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and card be the cardinality constraint. Let I_{fre} be the frequent intervals, such that for any $i \in I_{fre}$,*

- *$freI(L, card, i) \geqslant \tau$ for some threshold $\tau \in \mathbb{R}$, or*
- *$freI_{\%}(L, card, i) \geqslant \tau$ for some threshold $\tau \in [0, 1]$, where $freI_{\%}(L, card, i) = \frac{freI(L, card, i)}{\sum_{i' \in I_{Card}} freI(L, card, i')}$ provides the ratio of an interval $i \in I_{Card}$, or*

- $freI_{Entropy}(L, card, i) \geqslant \tau$ *for some threshold* $\tau \in [0,1]$*, where* $freI_{Entropy}(L, card, i) = p\frac{\sum_{i' \in I_{Card}} -p' \log_2(p')}{\log_2(|I_{Card}|)}$*,* $p = freI_{\%}(L, card, i)$ *and* $p' = freI_{\%}(L, card, i')$*.*

Definition 6.24 provides three solutions to identify frequent intervals by filtering infrequent intervals, i.e., noise, based on a threshold. According to the first solution, an interval is frequent if its frequency is equal to or larger than the given threshold (i.e., an integer). Figure 6.24 shows an example using the first solution to identify frequent intervals. With a threshold 1, the intervals (0), (1) and (2+) are frequent (highlighted in black). In this case, we do not really filter noise, since an interval is frequent only if it is observed. With a threshold 25, the intervals (1) and (2+) are frequent since their frequencies (57 and 30) are above the threshold. Similar to the method for identifying frequent variants in Section 6.2.2, it is also possible to filter noise based on a relative ratio or entropy.

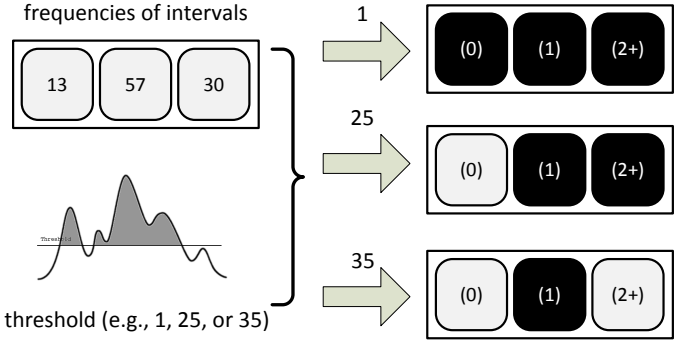


Figure 6.24: Identifying frequent intervals (colored in black) based on thresholds.

Definition 6.25 (Discovery of Precise Cardinalities by Frequent Intervals) *Let* $\{0\}$*,* $\{1\}$*,* $\{0,1\}$*,* $\{0,1,...\}$ *and* $\{1,2,...\}$ *(denoted as* 0*,* 1*,* 0..1*,* * *and* 1..**, respectively in the graph) be five normalized cardinalities. card is the cardinality constraint to be discovered and* I_{fre} *is its corresponding frequent intervals. The content of card, i.e.,* $conCard(card)$*, is discovered as*

- $\{0\}$ *if* $I_{fre} = \{(0)\}$*, or*
- $\{1\}$ *if* $I_{fre} = \{(1)\}$*, or*
- $\{0,1\}$ *if* $I_{fre} = \{(0),(1)\}$*, or*
- $\{1,2,...\}$ *if* $(0) \notin I_{fre} \wedge (2+) \in I_{fre}$*, or*

- $\{0, 1, ...\}$ *if* $I_{fre} \supseteq \{(0), (2+)\}$.

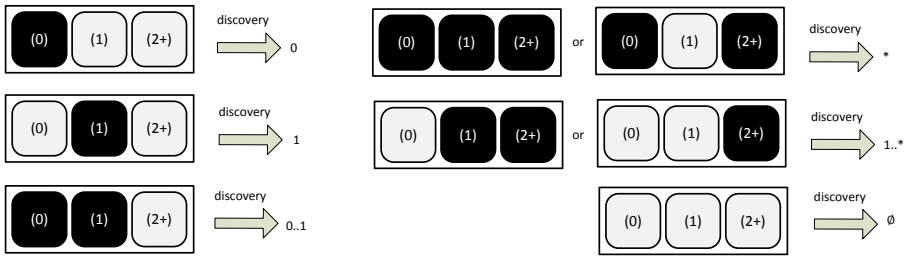


Figure 6.25: Discovery of normalized cardinalities by frequent intervals.

Definition 6.25 and Figure 6.25 show how to transform frequent intervals to normalized cardinalities by enumeration. Note that if there are no frequent intervals, which indicates that the threshold is configured too high, there are no corresponding normalized cardinalities.

6.3 Evaluation

In this section, we evaluate the OCBC model discovery approaches by comparing them with other model discovery approaches in terms of the same data from a business process. More precisely, we first extract event logs, i.e., XOC logs and XES logs, from the data generated in the OTC scenario of a real ERP system Dolibarr. Then different approaches are employed to discover various models from event logs. At last, we compare those discovered models and evaluate the ability of OCBC models to capture the real operational processes.

6.3.1 Business Process

The OTC (order-to-cash) business process is a typical and significant scenario in enterprises. It is supported by ERP systems, e.g., Dolibarr, an open source ERP system for small and medium enterprises. The OTC business process covers a range of operations from creating orders to paying bills on the behavioral (i.e., control-flow) perspective. Besides, operations may add, update or delete objects, e.g., records in database tables and documents in disks, on the data perspective. For simplicity, we choose four important activities, i.e., “create order”, “create

shipment", "create invoice" and "create payment" to represent the possible types of operations, and four classes, i.e., "order", "shipment", "invoice" and "payment" to represent the possible types of objects, in the process. Moreover, there exist various interactions or relationships between different entities in the process. Here, we classify them into three types, i.e., relationships (i) between classes, (ii) between activities and (iii) between classes and activities.

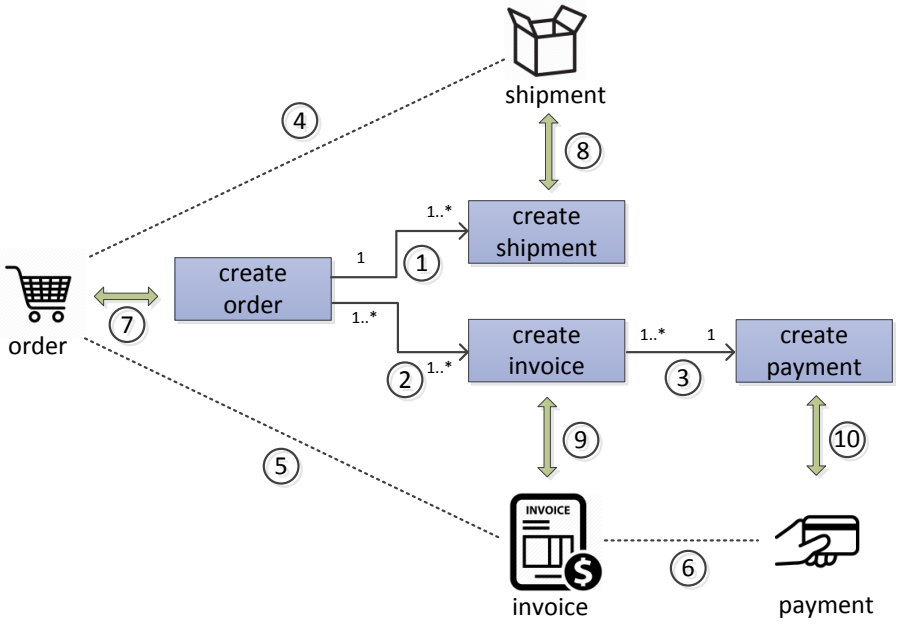


Figure 6.26: An informal model to describe the OTC business process.

Figure 6.26 employs an informal notation to describe the business process. The edges with single arrows indicate the temporal orders between activities while the cardinalities on edges indicate the restriction on the correspondences (e.g., one-to-many and many-to-many) between activities. For instance, edge ③ means that each "create payment" event must occur after its corresponding "create invoice" event, while the cardinalities on the edge means that each "create invoice" event corresponds to precisely one "create payment" event and each "create payment" event corresponds to one or more "create invoice" events. Dot-

ted lines are employed to show the relationships between classes. For instance, edge ⑤ refers to the relationship between "order" and "invoice". The interactions between the behavioral perspective and data perspective are denoted by edges with double arrows. For instance, "create invoice" interacts with "invoice" through edge ⑨.

Based on the data generated in the process described above, we extract XOC logs based on the approach in Chapter 4, resulting in the log in Table 6.1 (only a segment of the log is shown). In contrast, the method in [136] is used to extract XES logs, considering the order id as the case id. In order to make the comparison easier to understand, we only consider a segment of the whole logs for discovering models. For instance, Figure 6.27 shows two cases $case1$ and $case2$ in the XES log.

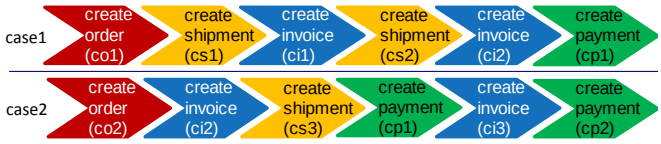


Figure 6.27: A segment of the extracted XES log.

Note that the generated XES log has (i) convergence problems, e.g., $ci2$ is contained by two cases as if it is executed twice though it is performed only once in Dolibarr, and (ii) divergence problems, e.g., in case $case2$ "create payment" has two instances $cp1$ and $cp2$ which cannot be distinguished in the case, though they are performed on different documents in the Dolibarr (i.e., $cp1$ is on $ci2$ and $cp2$ is on $ci3$).

6.3.2 Comparison of Discovered Models

Based on the generated XOC log, we can discover an OCBC model as shown in Figure 6.14. Next, we discover other models such as Petri nets, BPMN diagrams, Declare models and directly follow graphs. Then all the models are compared by checking which model can best describe the OTC business process.

Figure 6.28 displays a discovered data Petri net based on the approach in [33, 83]. On the behavioral perspective, the discovered Petri net has a lot of implicit transitions and loops. It is not precise and allows for behavior which is not possible in the real business process. For instance, the "create shipment"

activity can be skipped in the discovered Petri net, which violates the real process in its statement that each "create order" should be followed by at least one "create shipment" event. Besides, the many-to-many relationship between "create order" and "create invoice" in the process is transformed into a one-to-many relationship. On the data perspective, some variables and guards are discovered, which present the conditions to trigger transitions. The dotted line shows the interactions between activities and variables.

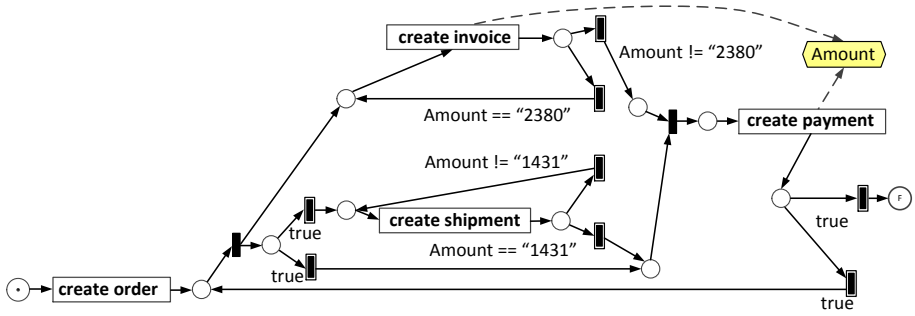


Figure 6.28: A data Petri net discovered by Inductive Miner (behavioral perspective) and Decision-Tree Miner (data perspective).

Figure 6.29 presents a discovered BPMN model using the approach in [30]. The model is not precise, e.g., the sub-process is a "flower" model involving activities "create payment" and "create invoice", which can happen at any order for any number of times. This approach only discovers the behavioral perspective with hierarchies, and we do not find any existing work which can automatically discover the data perspective. However, it is possible to manually add variables, data objects and data stores (onto the discovered model) to describe the data perspective based on expert knowledge.

A discovered data-aware Declare model [94] is shown in Figure 6.30(a). For simplicity, we limit the discovered constraint types to the set of "response", "not response", "precedence" and "not precedence". Declare models use a declarative way to express the constraints between any pair of activities [95]. For instance, the constraint $con1$ between "create invoice" and "create payment" indicates that each "create invoice" event should be followed by a "create payment" event and each "create payment" event should be preceded by a "create invoice" event. The real business process indicates that "create payment" events cannot occur

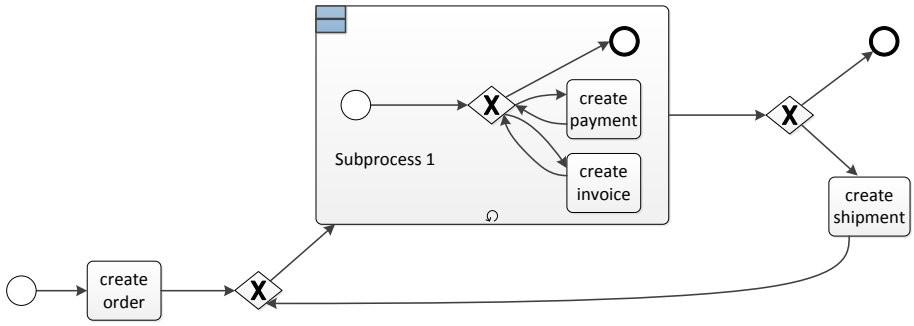
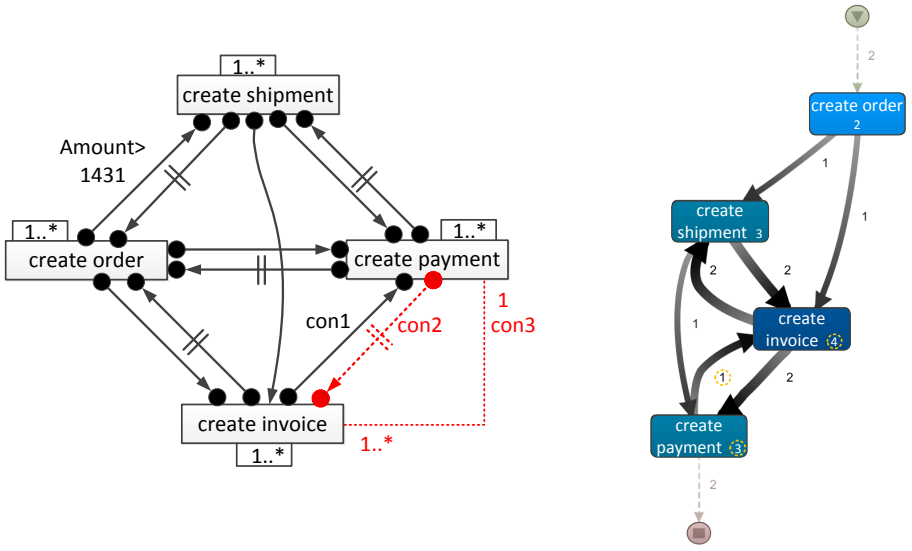


Figure 6.29: A model in the BPMN notation discovered by BPMN Miner.

before corresponding "create invoice" events. However, a negative constraint ($con2$) needed to describe this rule is not discovered, since in $case2$, the "create payment" event $cp1$ is wrongly related to its subsequent "create invoice" event $ci3$ (the divergence problem, cf. Chapter 4). Besides, existing constraint types in Declare models do not have a constraint type like $con3$ to restrict the cardinality between "create invoice" events and "create payment" events, which is required in the business process. The discovered data perspective consists of guards on constraints, which specify the situations when constraints are enabled.

A DFG presents the directly-follow relations between any pair of activities and possibly also shows the corresponding frequencies. Indicated by Figure 6.30(b), "create invoice" activity happens 4 times which violates the reality, i.e., 3 times (the convergence problem, cf. Chapter 4). Moreover, the DFG shows that a "create payment" event is directly followed by a "create invoice" event once, which also violates the rule in the real business process.

In comparison, our OCBC model shown in Figure 6.14 has stronger power to describe the real business process. On the behavioral perspective, since the XOC log has no convergence and divergence problems (multiple instances can be distinguished when we relate events based on correlation patterns, cf. Section 6.1.4), all constraints are correctly discovered. Besides, with the support of a class model and AOC relations, the cardinality constraints (e.g., one-to-many and many-to-many relationships) between activities can also be clearly described. Moreover, the data perspective is powerfully described with a class model. In the future, it is possible to discover the attributes of classes and activities, and guards (used in data Petri net) for constraints.



(a) A Declare model discovered by Declare Miner.

(b) A DFG discovered by Disco.

Figure 6.30: Two models discovered by Declare Miner and Disco.

6.4 Related Work

A process discovery technique takes an event log as an input and produces a model without using any a priori information. The model reveals how the business process is executed in reality. Currently, there exist approaches and tools in academia and industry to discover various imperative and declarative models. In this section, we discuss the related approaches.

6.4.1 Classical Process Discovery Techniques

Typically the focus of process discovery techniques is on the control-flow aspect of a process. The α-algorithm [149] is the first approach to discover a process model (a Petri net) from an event log to show the control-flow perspective. After that, various approaches such as heuristic mining [159, 160], fuzzy mining [58], region-based mining [12, 151], genetic process mining [18, 36], and inductive mining [80–82, 84] are proposed. The approaches discover different types of

models, such as Petri nets, BPMN diagrams and Declare models. These discovered models reveal insights of business processes from different angles. One of the existing state-of-the-art techniques is the visual inductive miner, which always returns a sound process model and is able to handle large and noisy event logs. Nevertheless, the miner can ensure (if desired) perfect fitness. Results can be converted to Petri nets, EPCs, state charts and BPMN models. Moreover, the visual inductive miner supports bottleneck analysis, replay animation and outlier detection.

There are also some data-aware process mining discovery techniques to discover models with data. [33,124] extend the control-flow perspective with the data perspective (e.g., read and write operations, decision points and transition guards) using standard data mining techniques. [99] proposes the Data-aware Heuristic Miner (DHM), a process discovery approach which uses the data attributes to distinguish infrequent paths from random noise by classification techniques. Data perspective and control-flow perspective of the process are discovered together. DHM is robust against random noise to some degree, and reveals data-driven decisions. These data-aware techniques can discover the data perspective to some extent, but they cannot discover a strong data perspective with the more powerful constructs (e.g., cardinality constraints) used in ER models and UML class models. Besides, they are still considered as case-centric, since they correlate events based on case notions.

6.4.2 Declarative Process Discovery Techniques

Besides discovering imperative models discussed above, there exist approaches in [94–96, 121, 122] (e.g., Declare Miner/Declare Maps Miner) which can discover Declare models, i.e., declarative constraints representing the actual behavior of a process as recorded in an event log.

Based on some a priori knowledge, [95] enables the selection of a list of Declare templates for the discovery task, and the approach returns a model only containing constraints (i.e., instantiations) of the selected templates. In this way, discovered constraints are the most interesting from the domain point of view, thus reducing the complexity of the resulting models. In order to deal with noise, one can specify thresholds for two parameters *minimum support* and *alpha*. For instance, the parameter *minimum support* decides the percentage of traces in which a constraint (to be discovered) must be satisfied.

[121] presents an approach to discover Declare models with hierarchies. It can reduce declarative model complexity by aggregating activities according to inclusion and hierarchy semantic relations. The idea is to employ natural

language processing to identify common objectives between activity labels, and then abstracts these activities into hierarchies based on Wordnet2.

[94] automatically discovers declarative process models that incorporate both control-flow dependencies (behavioral perspective) and data conditions (data perspective), by extending Declare with the ability to define data conditions (predicates). The data-aware Declare notation is defined using LTL-FO (First-Order LTL) rules. A rule indicates an association between an activity, a condition and another activity, e.g., if an activity is executed and a certain data condition holds after this execution, some other activity must eventually be performed.

There is also work [37, 62] to discover named DCR graphs which are also declarative models and similar to Declare models.

6.4.3 Artifact-Centric Discovery Techniques

A few discovery techniques [93, 109, 110, 117] have been developed to discover business processes in terms of so-called business artifacts. Artifacts have data and lifecycles attached to them, thus relating both perspectives.

The approach in [117] starts from a raw event stream and learns correlation information between the events to build an Entity-Relationship (ER) model. Guided by the user a so-called *artifact-centric log* is created. Such artifact-centric logs are used to discover the lifecycles of artifacts. Here any approach can be used to produce a Petri net. In the final step, the Petri nets are translated into the Guard-Stage-Milestone (GSM) notation [117]. The approach in [93] uses a different starting point. It first presents a semi-automatic and end-to-end approach to identify artifacts in a relational database of an ERP system. Then a life-cycle event log is extracted from event data stored in the database for each identified artifact. Finally, it learns a process model describing the process as a set of interacting data objects, i.e., artifacts (of a process), each following its own life-cycle.

Similar to the artifact-centric idea, [153] considers a single process having different facets, or perspectives, each with their own state space. It proposes an approach to discover state-based models, which concentrate on the states of different perspectives and explore how they interact with each other. More precisely, the notion of a Composite State Machine is defined as a way to model multi-perspective processes that can be learned from event logs. As the resulting models can be quite complex, three different operations are provided such that simplified views can be created on state machines.

6.4.4 Data Model (Data Schema) Discovery Techniques

The discovery approach for OCBC models proposed in this chapter involves the discovery of the data perspective, i.e., a class model. Here, we also discuss the literature related to the discovery of the data model or data schema in databases.

Most work in this field focuses on computing inclusion dependencies. An inclusion dependency over two predicates R and S from a schema is written as $R[A_1,...,A_n] \subseteq S[B_1,...,B_n]$, where A_i and B_i are distinct attributes (column names) of R and S. It implies that each tuple of values appearing in columns $A_1,...,A_n$ must also appear as a tuple of values in columns $B_1,...,B_n$. It is like that each value in a foreign key column must exist in its corresponding primary key column. [10] proposes an approach named SPIDER, for efficiently detecting single-column inclusion dependencies. It first sorts the distinct values in all columns and then uses a parallel merge-sort like algorithm to compute all inclusions simultaneously.

Based on the discovered inclusion dependencies, it is possible to detect foreign keys and foreign/primary key relationships between relational tables. The state-of-the-art algorithm for efficient and automatic discovery of single and multi-column primary keys is Gordian [129]. Gordian formulates the problem as a cube computation that corresponds to the computation of the entity counts of all possible column projections. [164] proposes a robust algorithm for discovering single-column and multi-column foreign keys, effectively reducing a large number of false positive pairs produced by partial inclusion.

Surprisingly, very little work has dealt with the discovery of cardinality constraints on the foreign/primary key relationships between tables. In our discovery approach, we mainly focus on this part, and provide solutions to deal with noise.

6.4.5 Tools for Discovery

Here, we list the available tools related to model discovery. ProM is the leading free and open-source process mining tool platform in academia. Currently, there are over 1500 plug-ins available (including deprecated plugins that are no longer supported). ProM supports dozens of process discovery algorithms which can load XES, MXML, and CSV files. The plugins related to the most known discovery approaches are shown in Table 6.13.

Other than the academic tools, there is also a variety of commercial software. Table 6.14 lists some process mining tools: Celonis Process Mining (Celonis), Disco (Disco), Enterprise Discovery Suite (EDS), Interstage Business Process

Name	Approach	Output Model
Alpha Miner	[149]	Petri net
BPMN Miner	[30]	BPMN diagram
CSM Miner	[153]	state-based model
Declare Maps Miner	[94–96, 122]	Declare model
ILP Miner	[155]	Petri net
Heuristics Miner	[99, 159]	Heuristics net
Inductive Visual Miner	[80–82, 84]	Petri net/Process tree
Episode Miner	[78]	Episode model
ETMd	[17]	Process tree

Table 6.13: Overview of the most known academic process mining discovery approaches in ProM.

Manager Analytics (Fujitsu), Minit (Minit), myInvenio (myInvenio), Perceptive Process Mining (Perceptive), QPR Process Analyzer (QPR), Rialto Process (Rialto), SNP Business Process Analysis (SNP), and web Methods Process Performance Manager (PPM). Tools like Disco, Fujitsu, QPR, and PPM have been around for a few years. Minit, myInvenio, and Rialto emerged very recently (in 2015).

Short name	Full name of tool	Vendor	Webpage
Celonis	Celonis Process Mining	Celonis GmbH	www.celonis.de
Disco	Disco	Fluxicon	www.fluxicon.com
Minit	Minit	Gradient ECM	www.minitlabs.com
QPR	QPR ProcessAnalyzer	QPR	www.qpr.com
...	...	...	...

Table 6.14: Overview of commercial process mining tools [136].

6.5 Summary

In this chapter, we propose some approaches to discover OCBC models, which describe behavioral (control-flow) perspective, data perspective and interactions in between in one diagram. These approaches can better deal with data with

many-to-many relations from databases of object-centric information systems. More precisely, they correlate events through the data perspective rather than a case notion, and can avoid the problems, such as incorrect constraints and frequencies due to convergence and divergence. Besides, it fills the complete separation between data/structure (e.g., a class model) and behavior (e.g., BPMN, EPCs, or Petri nets), which can not be done by existing discovery approaches.

In order to make the discovery approaches robust in real-life data with noise, we give some solutions to filter the noise and discover precise constraints, with configured thresholds on different perspectives. Based on a real business process, we discover OCBC models and other types of models from the same data generated in the process, and compare them. By comparison, the OCBC model can best describe the process in terms of data perspective and interactions.

There are many possible avenues for *future work*. In this chapter, we limit the input to an XOC log and the output to an OCBC model. It is possible to investigate discovering models directly on databases, which enables OCBC model discovery on big data. Besides, we can also consider discovering constraints involving the attributes of activities and classes, e.g., guards, for OCBC models. Moreover, cardinality constraints beyond normalized constraints can be discovered based on a probability distribution. A possible idea is that $k..n$ can be discovered if the cumulation of numbers from k to n is above a configured threshold.

Chapter 7

OCBC Conformance Checking

In Chapter 4, we defined XOC event logs to organize the data from object-centric information systems, and in Chapter 6, we proposed approaches to discover OCBC models from XOC logs. After illustrating XOC logs and OCBC models individually, this chapter looks at the situation where both a process model and an event log are given. The model may have been constructed by hand or may have been discovered (and adapted using some known knowledge). Conformance checking searches for the difference between the modeled behavior and the observed behavior by relating events in the event log to activities in the process model and comparing both. It is relevant for business alignment and auditing. For example, the event log can be replayed on top of the process model to find undesirable deviations suggesting fraud or inefficiencies.

This chapter is organized as follows. In Section 7.1, we motivate the conformance checking with object-centric behavioral constraints (i.e., OCBC conformance checking) by discussing the challenges suffered by conventional conformance checking approaches when they are applied to detect deviations related to the data perspective. Section 7.2 checks conformance on the data perspective, behavioral perspective and interactions in between, respectively. In order to measure (quantify) the global conformance, we refer to existing criteria, i.e., fitness, precision and generalization, and propose approaches to compute them in the context of an OCBC model and an XOC log in Section 7.3. Based on the criteria, an advanced discovery approach is introduced, which can discover models with customized criteria and deal with noise. In Section 7.4, we evaluate our conformance checking approach and compare it with other conformance

checking techniques. Section 7.5 reviews the related work while Section 7.6 concludes this chapter.

7.1 Motivation for OCBC Conformance Checking

Companies often have process models (explicit or implicit) to describe how their business processes should be executed. However, in some situations it is possible to violate the predefined processes. For example, in a hospital it must be possible to react to urgent situations and, therefore, the flexibility to diverge from the normal flow of actions is crucial. Besides, in flexible business environments, descriptive models (less strict than prescriptive models) are often used to capture the processes. These models usually give suggestions to indicate how to operate information systems and it is common that users do not totally follow these suggestions. For instance, the reference models in the context of SAP R/3 and ARIS describe the "preferred" way processes should be executed. People actually using SAP R/3 may deviate from these reference models [125].

Revealed by the above discussion, there may exist conformance problems between predefined processes and real executions. For companies, it is necessary to know where the real behavior violates the predefined process and get rid of the violations when they correspond to unexpected situations. For instance, in financial systems, it is desirable to keep the actual procedure consistent with the model. The information systems employed to support the business executions allow diagnosing these conformance problems. More precisely, on the one hand, these systems typically log events (e.g., in transaction logs or audit trails) related to the actual business process executions. On the other hand, explicit process models describing how the business process should (or is expected to) be executed are frequently available (based on domain knowledge or discovery techniques). Based on them, conformance problems are identified as the disagreement between the logs and the models.

Various conformance checking techniques (and similar techniques such as compliance checking, auditing, Six Sigma, etc.) are proposed to diagnose the conformance problems, i.e., if reality, as recorded in the log, conforms to the model and vice versa. Most of them are based on replaying the traces in the event log on the model. In [125] the numbers of missing and remaining tokens are counted while replaying the event log, which is used to diagnose conformance problems on the control-flow perspective. State-of-the-art techniques are the so-called alignment-based approaches [7,137]. Given a trace in the event log, the closest path in the model is computed by solving an optimization problem. Based

on these techniques, it is possible to quantify the level of conformance. Most techniques focus on fitness, however, there are also techniques for computing other quality dimensions such as simplicity, precision, and generalization [104].

However, the existing conformance checking techniques often employ process-centric models, e.g., Petri nets, which typically consider process instances in isolation (ignoring interactions among them) and are more focused on the behavioral perspective of processes (cf. Chapter 5). Accordingly, when applying existing techniques on the object-centric systems such as ERP and CRM (which generate and store data in an object-centric manner), the diagnosis results are only on the behavioral perspective and fail to deal deviations related to the data perspective and interactions. Therefore, some useful insights (related to the undiscovered deviations) cannot be revealed.

Since the deviations related to the data perspective and interactions are important, in this chapter we employ OCBC models for conformance checking. Unlike existing approaches, instances are not considered in isolation and cardinality constraints in the data/object model are taken into account. Hence, we can now detect and diagnose a range of conformance problems that would have remained undetected using conventional process-model notations. More precisely, we abstract a set of rules representing an OCBC model and detect nine types of conformance problems. Besides, we define metrics such as fitness, precision and generalization which can be used to quantify the conformance between an OCBC model and an XOC log.

7.2 Diagnosing Conformance

Given an XOC event log L and an OCBC model M, we check whether reality (in the form of events and object models in L) conforms to the model M in this section. More precisely, the conformance diagnosis is split into three parts, separately: on the data perspective, the behavioral perspective and the interactions between them. In the diagnosis, nine types of conformance problems are identified. Most of these problems are not captured by existing conformance checking approaches [34, 43, 125, 137].

7.2.1 Conformance Checking on Data Perspective

The data perspective serves as the backbone of an OCBC model. Therefore, we first introduce how to check the conformance on the data perspective, i.e., check if each object model conforms to the class model. In Chapter 5, we defined

valid and fulfilled object models for a class model, which are used to check the conformance on the data perspective.

Definition 7.1 (Conformance on Data Perspective) *Let* $L = (E, act, attrE, relate, om, \preceq)$ *be a sound XOC event log and* $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ *be an OCBC model, where ClaM is a class model. Event log* L conforms *to OCBC model* M *on the data perspective if and only if the following rules are satisfied:*

- ***Each object model is valid (Rule I):*** *for any* $e \in E$*, object model* $OM_e = (Obj_e, Rel_e, class_e)$ *is valid for ClaM (this includes checking the* $\Box$ *cardinality constraints that should always hold as stated in Chapter 5), and*
- ***The last object model is fulfilled (Rule II):*** *for the last event* $e_l \in E$*, object model* $OM_{e_l} = (Obj_{e_l}, Rel_{e_l}, class_{e_l})$ *is fulfilled for ClaM (this involves checking the* $\Diamond$ *cardinality constraints that should eventually hold as stated in Chapter 5).*

Definition 7.1 checks the conformance on the data perspective according to two rules: (i) each object model is valid for the class model, which mainly requires that each object should satisfy the corresponding "always" cardinality constraints, and (ii) the last object model is fulfilled for the class model, which restricts that each object should satisfy the corresponding "eventually" cardinality constraints. Note that Rule I assigns restrictions on each object model and Rule II only assigns restrictions on the last object model.

Figure 7.1 shows how to check conformance on the data perspective. The left part is a class relationship between class *order line* and class *delivery*, with four cardinality constraints "$\Box$ 1..*", "$\Diamond$ 1..*", "$\Box$ 0..1" and "$\Diamond$ 1". Since the object model is evolving while the process is executed, the right part depicts two snapshots of the object model (we only present the objects of *order line* and *delivery* for simplicity) in an XOC log. The first snapshot corresponds to the moment t_i, including three *order line* objects ($ol1$, $ol2$ and $ol3$) and three *delivery* objects ($d1$, $d2$ and $d3$). t_n represents the last moment and the corresponding snapshot contains four *order line* objects ($ol1$, $ol2$, $ol3$ and $ol4$) and four *delivery* objects ($d1$, $d2$, $d3$ and $d4$).

In the first snapshot, object $ol2$ has two corresponding *delivery* objects ($d1$ and $d2$), thus violating the "$\Box$ 0..1" annotation. Note that object $ol2$ has one corresponding *delivery* object ($d2$) in the second snapshot, satisfying the "$\Diamond$ 1" annotation. Object $d3$ has never a corresponding *order line* object, thus violating the "$\Box$ 1..*" and "$\Diamond$ 1..*" annotations. Object $ol4$ has no corresponding *deliver* objects at the last moment t_n, thus violating the "eventually" cardinality constraint "$\Diamond$ 1".

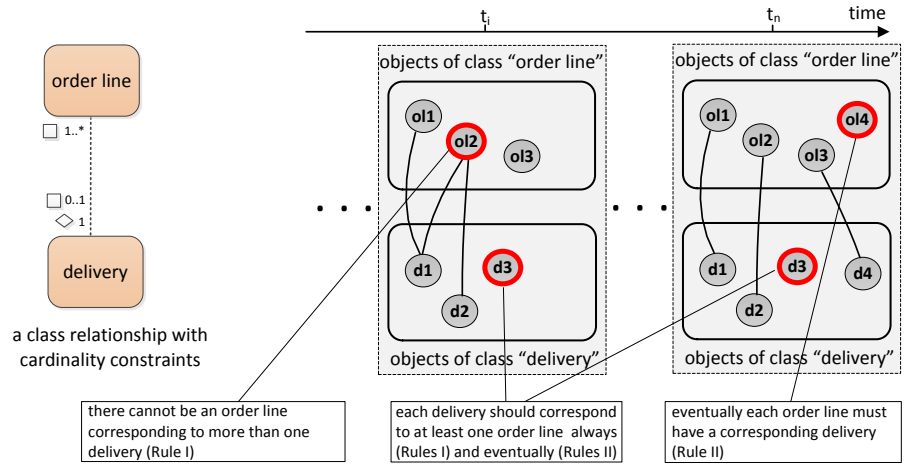


Figure 7.1: An illustration for checking conformance on the data perspective (detecting violations of data cardinality constraints).

7.2.2 Conformance Checking on Behavioral Perspective

Based on the data perspective, we can correlate events in XOC logs, resulting in pattern instances. Conformance checking on the behavioral perspective compares the correlated instances with the activity model.

Definition 7.2 (Conformance on Behavioral Perspective) *Let L be a sound XOC event log and $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model, where $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ is an activity model. Event log L* conforms *to OCBC model M on the behavioral perspective if and only if the following rules are satisfied:*

- ***Each activity exists (Rule III):*** *$\{act(e) \mid e \in E\} \subseteq A$, i.e., all activities referred to by events exist in the activity model, and*
- ***Each constraint is respected (Rule IV):*** *for each constraint $con \in Con$:*

  $$\forall ins \in extI(L,P) : ins_{\downarrow P} \in type(con),$$

  where $P = (\pi_{ref}(con), \pi_{tar}(con), crel(con))$ is the correlation pattern corresponding to con.

Definition 7.2 checks the conformance on the behavioral perspective according to two rules: (i) all activities observed in the log exist in the activity model, and (ii) all behavioral constraints are respected by correlated instances in the log. In OCBC models, each constraint con has a corresponding correlation pattern P. Indicated by Rule IV, con is respected if all the instances correlated by P (i.e., $extI(L,P)$), satisfy the semantics of con.

Rule III is relatively easy to understand since we only need to check events individually. In comparison, Rule IV involves a set of correlated events and is more difficult. Figure 7.2 shows how to check conformance on the behavioral perspective. The left part is a behavioral constraint of "unary-precedence" type between activities $a1$ and $a2$. The constraint corresponds to a correlation pattern $(a2, a1, r)$, i.e., $a2$ is the reference activity, $a1$ is the target activity and r is the intermediary. The right part presents three corresponding instances, in which $e1$, $e5$ and $e7$ are of activity $a2$, i.e., reference events, and the other events are target events. In each instance, the events are executed in the order indicated (from left to right).

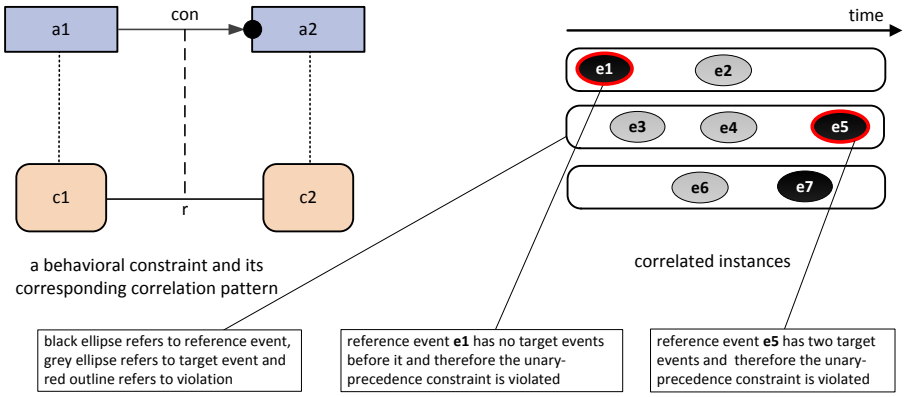


Figure 7.2: An illustration for checking conformance on the behavioral perspective (detecting violations of behavioral constraints).

For the constraint depicted in Figure 7.2: $type(con) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 1\}$, i.e., there should be precisely one target ($a1$) event preceding each reference ($a2$) event in each instance. Consider for example the third instance, in which $e7$ is the reference event and $e6$ is a target event occurring before $e7$. Hence, no problem is discovered for $e7$. In contrast, in the first instance there is

no target events occurring *before* the reference event $e1$, which signals a violation of constraint con. For the reference event $e5$ in the second instance, there are two target events $e3$ and $e4$, which also violates con since the number "two" is not allowed by the constraint.

7.2.3 Conformance Checking on Interactions

The interactions between the data perspective and the behavioral perspective are specified by AOC relationships. They indicate the allowed correspondences between events and objects. In this section, we check the conformance on the interactions, i.e., compare the observed reference relations (between events and objects) in an XOC log with the AOC relationships in an OCBC model.

Definition 7.3 (Conformance on Interactions) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model. O_L is the set of all objects in L, i.e., $O_L = \{o \mid \exists e \in E : o \in Obj_e\}$. Event log L* conforms *to the OCBC model M in terms of the interactions between data perspective and behavioral perspective if and only if the following rules are satisfied:*

- ***Each object "always" has the right number of events (Rule V):*** *for any $aoc = (a, c) \in AOC$ and $o \in \partial_c(O_L)$:*

  $$\forall e \in E_o : |corAOC(o, (a, c), e)| \in \sharp_A^{\Box}(aoc),$$

  where $E_o = \{e \in E \mid o \in Obj_e\}$ represents all moments when o exists and $corAOC(o, (a, c), e)$ returns all events correlated to the object o by (a, c) at the moment that e happens (cf. Chapter 6).

- ***Each object "eventually" has the right number of events (Rule VI):*** *for any $aoc = (a, c) \in AOC$ and $o \in \partial_c(O_L)$:*

  $$|corAOC(o, (a, c), e_l)| \in \sharp_A^{\Diamond}(a, c), if\ o \in Obj_{e_l},$$

  where e_l represents the last moment.

- ***Each event refers to objects of related classes (Rule VII):*** *for any $e \in E$ and $o \in relate(e)$:*

  $$(act(e), class_e(o)) \in AOC.$$

- ***Each event refers to the right number of objects (Rule VIII):*** *for any* $aoc = (a, c) \in AOC$ *and* $e \in \partial_a(E)$*:*

$$|\{o \in relate(e) \mid class_e(o) = c\}| \in \sharp_{OC}(aoc).$$

An event log L that satisfies the four rules defined above is conforming to M in terms of the interactions between the data perspective and the behavioral perspective:

- Rule V indicates that for each object, the number of its related events correlated by an AOC relationship aoc satisfies the "always" cardinality constraint, i.e., $\sharp_A^{\Box}(aoc)$, at each moment. Note that an object o may not exist at some moment. Therefore, we limit "each moment" to any moment when o exists.
- Rule VI only gives the restriction on objects at the last moment. More precisely, for each object o in the last object model, the number of its related events correlated by an AOC relationship aoc should satisfy the "eventually" cardinality constraint, i.e., $\sharp_A^{\Diamond}(aoc)$, at the last moment (i.e., e_l).
- Rule VII requires that the reference relations (between events and objects) should be consistent with AOC relationships. If an event e refers to an object o, there should exist an AOC relationship between the activity of e and the class of o, i.e., $(act(e), class_e(o)) \in AOC$. Otherwise, this rule is violated.
- Rule VIII implies that each event needs to have the right number of corresponding objects, correlated by an AOC relationship. For an event e, an object o is correlated to e by an AOC relationship (a, c), if o is a c object (i.e., $class_e(o) = c$) and is referred to by e, i.e., $o \in relate(e)$. Unlike "always" and "eventually" cardinality constraints, this rule gives restrictions on each event at the moment that the event happens.

Rule VII is relatively easy to understand. The other rules are related to the more complex interplay between events and objects over time. These are more interesting, but also more difficult to understand. Therefore, we elaborate on violations of Rules V, VI and VIII next, as shown in Figure 7.3. The left part is an AOC relationship between activity *pay* and class *ticket*, with three cardinality constraints "□ 0..1", "◇ 1" and "1..*". Since the object model is evolving while the process is executed, the right part depicts two snapshots of the process (we only present the events of activity *pay* and objects of class *ticket* for simplicity) in an XOC log. The first snapshot corresponds to the moment t_i, including three

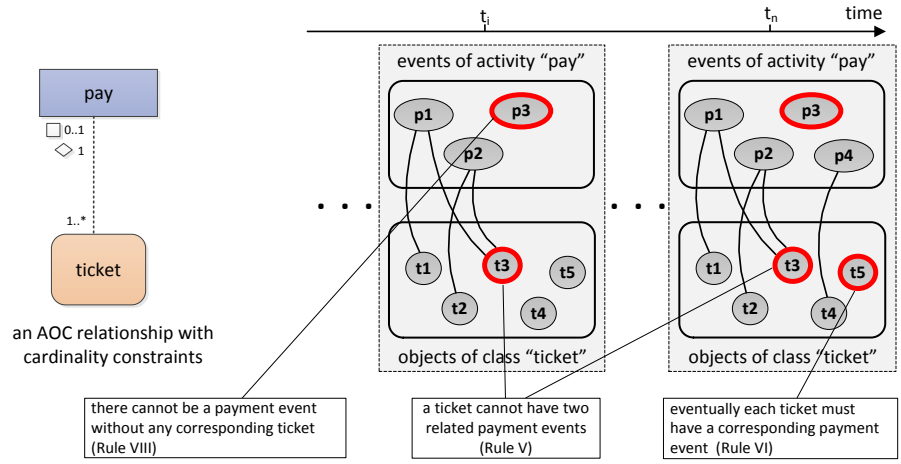


Figure 7.3: An illustration for checking conformance on interactions (detecting violations of cardinality constraints on AOC relationships between activities and classes).

events ($p1$, $p2$ and $p3$ which happens before t_i) and five objects ($t1$, $t2$, $t3$, $t4$ and $t5$ which exist at the moment t_i). t_n represents the last moment and the corresponding snapshot contains a new event $p4$ which refers to object $t4$.

Both in the first and second snapshots, object $t3$ has two corresponding *pay* events ($p1$ and $p2$), thus violating the "□ 0..1" annotation (Rule V). Object $t5$ has no corresponding *pay* events in the second snapshot (i.e., at the last moment), thus violating the "♢ 1" annotation (Rule VI). Note that object $t4$ has no corresponding *pay* events at moment t_i, but has one corresponding *pay* event $p4$ at the last moment, which does not violate the "eventually" cardinality constraint "♢ 1". Event $p3$ has no corresponding *ticket* objects, thus violating the "1..*" annotation according to Rule VIII. Note that the objects referred to by an event are fixed when the event happens. Therefore, if an event violates the Rule VIII at some moment t_i, the event is violating from that moment t_i to the last moment t_l, e.g., $p3$ in Figure 7.3.

7.2.4 Diagnostics Results

The conformance checking rules (Rules I,II...,VIII) illustrated above characterize a wide range of conformance problems, i.e., deviations, between a given model

and a log. According to the eight rules, we identify eight corresponding types of deviations. As shown in Figure 7.4, besides reporting deviations of various types, we also map deviations onto the log (resulting in a log view) and the model (resulting in a model view). By correlating deviations to elements in XOC logs and OCBC models, i.e., highlighting deviations in the log view and model view, it helps to reason about the sources which lead to these deviations.

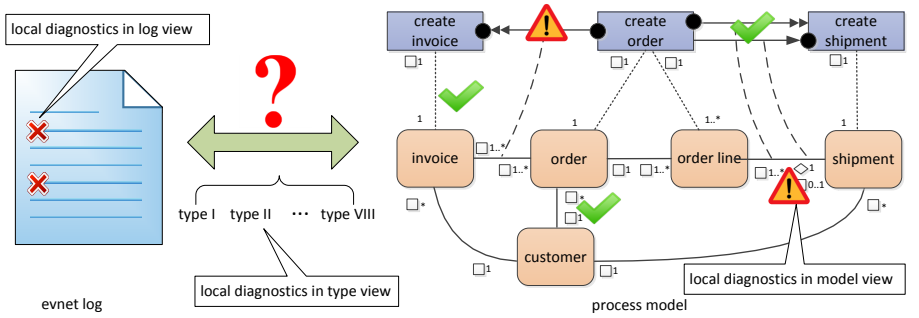


Figure 7.4: Presenting the diagnosis result in three views: type, log and model views.

Figure 7.4 shows the idea to present the conformance checking result in three views. Here, the three views consist of various local diagnostics, i.e., the whole conformance checking result is divided into different parts. Users can choose the part which they are interested in, and "zoom in" on the details related to the part. For instance, if one is interested in a violated behavioral constraint, the deviations related to the constraint will be presented when the constraint is selected. In the next section, we propose some criteria which quantify the conformance on the global level.

Next, we explain how to highlight the deviations of each type in log and model views that are summarized in Table 7.1:

- **Diagnostics for Type I deviations (validity of object models):** In the log view, each event corresponding to an invalid object model is highlighted. For each highlighted event, the elements (objects and object relations) which make the object model invalid are highlighted. In the model view, the violated □ cardinality constraints on class relationships are highlighted.
- **Diagnostics for Type II deviations (fulfillment):** In the log view, elements (objects and object relations) which make the last object model unfulfilled are highlighted. In the model view, the violated ♢ cardinality

constraints on class relationships are highlighted.

- **Diagnostics for Type III problems (activity existence):** In the log view, events whose activities do not exist in the activity model are highlighted.
- **Diagnostics for Type IV deviations (behavioral constraints are respected)**: In the log view, each deviating reference event (which does not have the right number of target events before and after it) is highlighted. In the model view, the violated behavioral constraints are highlighted.
- **Diagnostics for Type V deviations ("always" right number of events per object):** In the log view, each deviating object (which does not always have the right number of corresponding events) is highlighted. In the model view, the violated □ cardinality constraints on AOC relationships are highlighted.
- **Diagnostics for Type VI deviations ("eventually" right number of events per object):** In the log view, each deviating object (which does not eventually have the right number of corresponding events) is highlighted. In the model view, the violated ♢ cardinality constraints on AOC relationships are highlighted.
- **Diagnostics for Type VII problems (proper classes):** In the log view, events that refer to classes they should not refer to are highlighted.
- **Diagnostics for Type VIII deviations (right number of objects per event):** In the log view, each deviating event (which does not have the right number of corresponding objects) is highlighted. In the model view, the violated cardinality constraints (on the class side) on AOC relationships are highlighted.

Type	Highlighted elements	
	Log view	Model view
I	objects and object relations resulting in invalidity	□ cardinalities on class relationships
II	objects and object relations breaking fulfillment	♢ cardinalities on class relationships
III	events of inexistent activities	-
IV	events having incorrect numbers of target events	behavioral constraints
V	objects not always having correct numbers of events	□ cardinalities on AOC relationships
VI	objects not eventually having correct numbers of events	♢ cardinalities on AOC relationships
VII	events referring to objects of wrong classes	-
VIII	events referring to incorrect numbers of objects	cardinalities on the class side on AOC relationships

Table 7.1: Highlighting deviations of each type with the log and model views.

The approach which detects these various deviations is very different from existing conformance checking approaches. Most of the conformance checking approaches [34, 43, 125, 137] only consider control-flow and are unable to

uncover the above problems. Recently, conformance checking approaches based on alignments have been extended to also check conformance with respect to the data perspective [34, 98]. However, these do not consider a data model and focus on one instance at a time.

Interestingly, conformance over OCBC models can be checked very efficiently. In particular, each of the eight rules can be formalized as a boolean, SQL-like query over the input log. The final result is obtained by conjoining all the obtained answers.

7.3 Quantifying Conformance

In Section 7.2, we checked the conformance between an OCBC model and an XOC log by detecting deviations based on the eight rules extracted from the model. Besides the local conformance diagnosis, it is possible to quantify the global conformance based on criteria. In this section, we focus on the global conformance measure and give approaches to compute such criteria.

7.3.1 Basic Idea of Criteria

Quantifying the conformance between a log and a model is difficult, since it is characterized by many dimensions. In traditional process mining, fitness, simplicity, precision, and generalization are used to evaluate the quality of a model discovered from a log. Actually, these criteria also reveal the conformance between the log and the discovered model, since a discovered model has good quality if it conforms to the log. Note that among these criteria, the simplicity dimension only indicates the complexity of the discovered model, and it is not related to behavior in the log. Therefore, in this section, we abstract from simplicity and use fitness, precision and generalization to quantify the conformance.

The conformance diagnosis in Figure 7.4 gives a local diagnosis based on eight rules. In contrast, in this section, we measure the global conformance between a log and a model through fitness, precision and generalization, as shown in Figure 7.5. Note that these criteria only focus on the behavioral perspective, which are explained as follows:

- fitness: the model should allow for the behavior seen in the event log,
- precision: the model should not allow for behavior completely unrelated to what was seen in the event log, and
- generalization: the model should generalize the behavior seen in the event log.

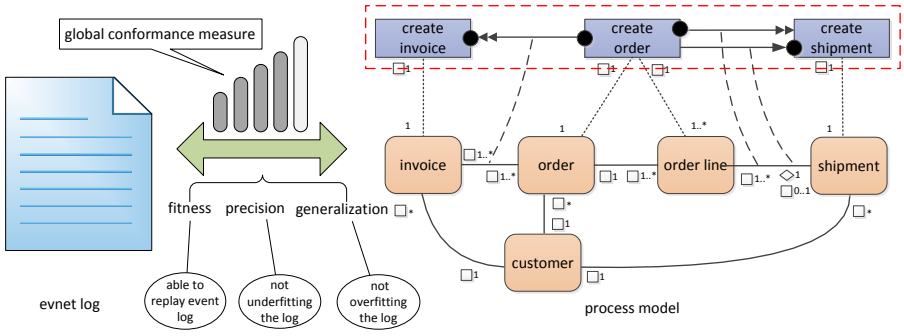


Figure 7.5: Quantifying the conformance on the behavioral perspective through three criteria: fitness, precision and generalization.

In traditional process mining, the replay [35, 123, 125] techniques are used to compute fitness between a Petri net and an XES log. By replaying the log on the model, i.e., replaying each case to the places and transitions, four counters are employed to count produced tokens, consumed tokens, remaining tokens and missing tokens. Based on these four numbers, fitness is derived to quantify the degree of fitting (cf. Section 7.5). Besides, alignment [3, 5–7] techniques are more advanced approaches to calculate fitness. For each case in the log, a corresponding model path (which is the closest to the case) is derived. Then the log moves (impossible in the model) and model moves (not observed in the log) are counted and assigned with cost. Finally, a value is derived based on the total cost to quantify the fitness.

Unlike traditional process mining, we do not assume a global case notion for the whole process in the context of an OCBC model and an XOC log. Each behavioral constraint in an OCBC model corresponds to a correlation pattern and specifies a restriction on events in a scope (rather than a case) identified by the pattern. In other words, a behavioral constraint is defined in the context of a correlation pattern, such that constraints corresponding to different patterns are independent. Because of this, it is not necessary to check an XOC log on the whole OCBC model (with all activities and constraints). We only need to *check conformance pattern by pattern*. The existing replay and alignment techniques assume case notions and check conformance on the whole process, which does not apply to our situation. For instance, it makes no sense to replay a pattern instance (i.e., an instance correlated by a correlation pattern) on the whole

process or find a model path, since the instance only has events of two activities.

Due to the difference discussed above, we employ a different methodology from traditional process mining techniques to compute the criteria. Based on the idea of *checking conformance pattern by pattern*, we (i) split the task of quantifying conformance on the whole process into several tasks of quantifying conformance on all correlation patterns in the process, and (ii) integrate the results of all correlation patterns into the whole criteria. As shown in Figure 7.6, based on a correlation pattern, we correlate events in the log to derive a set of pattern instances and extract all behavioral constraints corresponding to the pattern from the model. The criteria on the pattern level are computed for each pattern and at last they are merged as criteria on the model level.

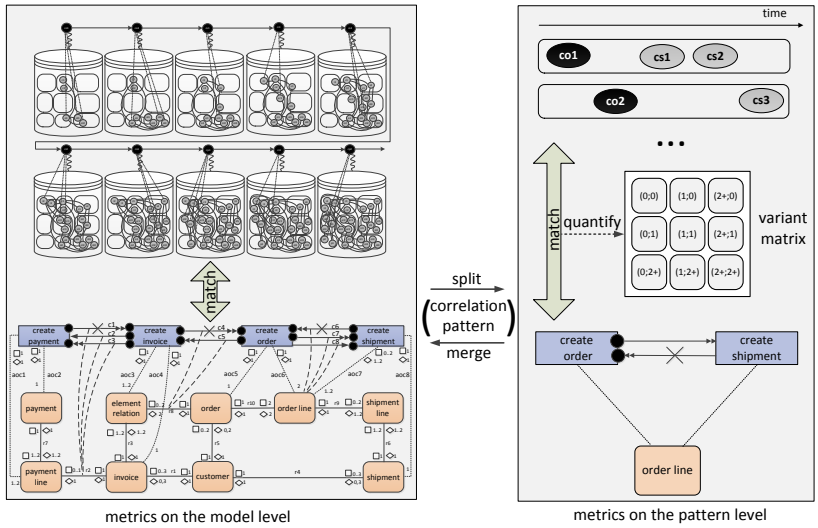


Figure 7.6: The task of quantifying the conformance on the model level can be split into several tasks on the pattern level.

In order to compute the criteria on the pattern level, we need to somehow connect the instances to the constraints, and then quantify the extent to which the instances match the constraints. In the context of a correlation pattern, the constraints can be considered as the allowed numbers of target events before and after each reference event, and the instances can be considered as the observed numbers of target events before and after each reference event. Intuitively, the variant matrix defined in Chapter 6 can serve as a bridge to connect the instances

to the constraints, since each variant in the matrix specifies a set of number pairs, in which the first/second number counts the target events before/after the reference event. The details are illustrated in the next section.

7.3.2 Connecting Event Log to Process Model

In general, the conformance between a log and a model indicates how much the observed behavior conforms to the allowed behavior. In order to quantify the conformance, we have to connect the log to the model. Based on the idea in Figure 7.6, the task of quantifying the conformance on the model level can be split into several tasks on the pattern level. Accordingly, the task of connecting the log to the model is converted into the task of connecting instances to behavioral constraints in the context of a correlation pattern.

The variant matrix V_{CT} defined in Chapter 6 can serve as a bridge to connect instances to constraints. More precisely, we map the instances onto the variant matrix to identify the observed variants, and map the constraints onto the variant matrix to identify the allowed ones. Then the criteria are computed based on the relation between observed and allowed variants. Next, we explain how to derive the observed variants and their frequencies.

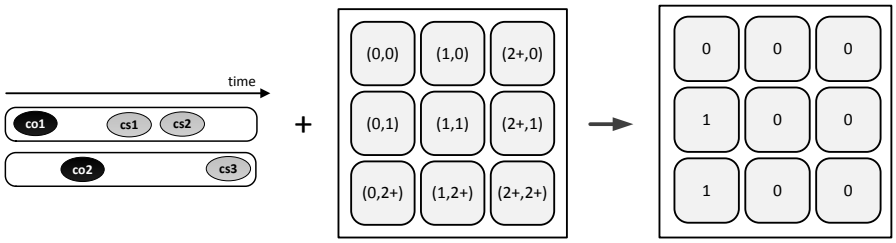


Figure 7.7: Function *freV* mapping instances (correlated by a correlation pattern) onto the variant matrix V_{CT} to show observed behavior, resulting in a *frequency matrix*. For instance, the first one of the instances (on the left side) only has two target events *cs1* and *cs2* after the reference event *co1*. This relation satisfies the semantics of the cell $(0;2+)$ in the variant matrix (in the middle). Since only one of the instances has such relation, the value in the cell corresponding to $(0;2+)$ in the frequency matrix (on the right side) is 1.

Among the instances correlated by a candidate pattern, if the relation between a reference event and its target events in some instance satisfies the semantics of

a variant, we say that the instance corresponds to the variant and the variant is observed once in the log. Based on this idea, a function $freV \in \mathcal{U}_L \times \mathcal{U}_P \times V_{CT} \rightarrow \mathbb{N}$ is defined (in Chapter 6) to compute how many times each variant is observed in a log corresponding to a pattern. More precisely, the frequency of a variant is equal to the number of instances corresponding to the variant. For instance, assume that we have two instances $\{\langle co1, cs1, cs2\rangle, \langle co2, cs3\rangle\}$ where $co1$ and $co2$ are reference events in Figure 7.7. For the first instance, there are zero and two target events before and after the reference events, respectively, i.e., $\langle co1, cs1, cs2\rangle_{\downarrow P} = (0,2) \in (0;2+)$. Similarly $\langle co2, cs3\rangle_{\downarrow P} = (0,1) \in (0;1)$. Therefore $freV(L,P,(0;2+)) = 1$ and $freV(L,P,(0;1)) = 1$.

In the variant matrix, each variant essentially is a constraint type, e.g., $(2+;0) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before \geq 2\}$. In an OCBC model, each behavioral constraint corresponds to a constraint type. Consider for example the constraint $con8$ in in Figure 7.8. $type(con8) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after \geq 1\}$. Therefore, constraints can be related to the variant matrix in terms of constraint types. Next, we define a function to map constraints (corresponding to a correlation pattern) onto a variant matrix.

Definition 7.4 (Allowed Variants by Model) *Let $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model where $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ is an activity model, and $P = (a_{ref}, a_{tar}, cr)$ is a correlation pattern.*

Function $posV \in \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow \mathbb{P}(V_{CT})$ returns the variants allowed by a model corresponding to a correlation pattern such that $posV(M,P) = \{v \in V_{CT} \mid \forall con' \in Con_P : v \subseteq type(con')\}$ where $Con_P = \{con \in Con \mid \pi_{ref}(con) = a_{ref} \wedge \pi_{tar}(con) = a_{tar} \wedge crel(con) = cr\}$.

In terms of a correlation pattern P, function $posV$ maps the constraints corresponding to P from an OCBC model M onto the variant matrix, resulting in a colored matrix. In the colored matrix, the black variants represent the behavior allowed by the model. Assume that $con6$ and $con8$ in Figure 7.8 are all constraints corresponding to the pattern $P = (create\ order, create\ shipment, order\ line)$ from the OCBC model M. Since $con6$ is of the *non-precedence* constraint type and $con8$ is of the *response* constraint type, $type(con6) \cap type(con8) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid before = 0 \wedge after \geq 1\}$. After checking all variants in the matrix, we get $(0;1) \subseteq (type(con6) \cap type(con8))$ and $(0;2+) \subseteq (type(con6) \cap type(con8))$. Therefore, $posV(M,P) = \{(0;1),(0;2+)\}$.

In traditional process mining, a log and a model are connected by replaying each case in the log onto the model. Since XOC logs and OCBC models do not assume a case notion, we employ a different approach, i.e., the variant matrix, to

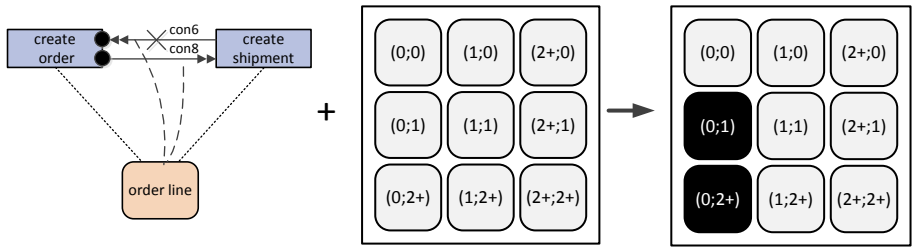


Figure 7.8: Function *posV* mapping constraints (for a correlation pattern) onto the variant matrix to show allowed behavior, resulting in a *colored matrix*.

connect a log to a model. Up to now, we have proposed functions to map an XOC log on the variant matrix (resulting in a frequency matrix to show the observed variants) and to map an OCBC model onto the variant matrix (resulting in a colored matrix to show the allowed variants). Next, we connect the XOC log to the OCBC model by overlapping the frequency matrix and the colored matrix.

Definition 7.5 (Connecting XOC Log to OCBC Model) *Function posV indicates a colored matrix which represents the allowed behavior, and function freV indicates a frequency matrix which represents the observed behavior. By overlapping and aligning these two matrices, an overlapped matrix is generated, which connects an XOC log to an OCBC model*

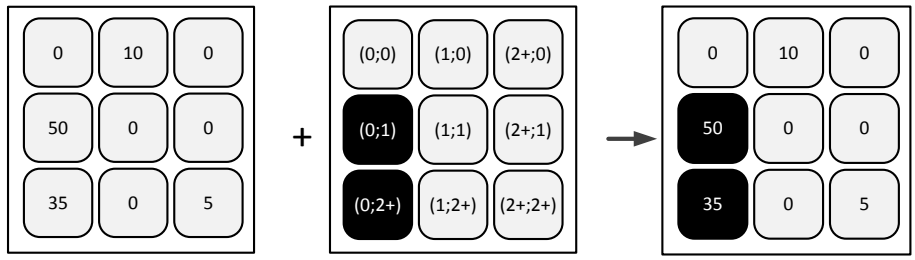


Figure 7.9: A log is connected to a model by overlapping the frequency matrix and the colored matrix, resulting in an *overlapped matrix*.

By overlapping the frequency matrix and the colored matrix, we derive an *overlapped matrix* which contains all the information (i.e., the allowed behavior and observed behavior) from these two matrices. Note that the positions of the

nine variants are fixed in all matrices, e.g., the bottom left cell corresponds to variant $(0;2+)$ in the frequency matrix, the colored matrix and the overlapped matrix. In Figure 7.9, the left matrix is a frequency matrix, which indicates the variants observed in the log, e.g., $(0;2+)$ is observed thirty-five times. The middle matrix is a colored matrix, which indicates that variants $(0;1)$ and $(0;2+)$ are allowed by the model. By overlapping them, we get the right matrix, i.e., an overlapped matrix. The overlapped matrix indicates that $(0;1)$ and $(0;2+)$ are observed fifty and thirty-five times, respectively, and they are allowed by the model. In contrast, $(1;0)$ and $(2+;2+)$ are observed ten and five times, respectively, and they are not allowed by the model. By summarizing the information in the overlapped matrix, we can claim that most observed behaviors are allowed by the model.

Through the variant matrix, we make a bridge to connect the log to the model, resulting in an overlapped matrix which indicates both allowed behavior and observed behavior. Next, we illustrate how to compute fitness, precision and generalization, based on the overlapped matrix.

7.3.3 Fitness

The most dominant question in the context of conformance is whether the real business process complies with the specified behavior, i.e., whether the log fits the model. In traditional process mining, a model with a perfect fitness can replay all traces in the log from beginning to end, and a model with a poor fitness allows for little behavior seen in the event log. Based on this idea, the fitness is often expressed as a value between 0 (very poor fitness) and 1 (perfect fitness).

In this section, we refer to the idea of fitness in traditional process mining and propose solutions to compute fitness in the context of XOC logs and OCBC models. More precisely, we quantify the fitness as the extent to which the observed variants (in the XOC log) consist to the allowed variants (by the OCBC model) based on the overlapped matrix derived in Section 7.3.2.

Figure 7.10 shows three overlapped matrices indicating different fitting situations. Intuitively, the first situation has a perfect fitness, since all observed behavior in the log is allowed by the model. Additionally, the second overlapped matrix shows an almost fitting situation, since most observed behavior is allowed. Differently, only one observed variant is allowed in the third overlapped matrix, describing a non-fitting situation.

According to the above discussion, we explain our approach to compute fitness. In general, fitness can be expressed as the ratio of the observed and allowed variants in all observed variants. Based on this idea, the fitness is 1 if all

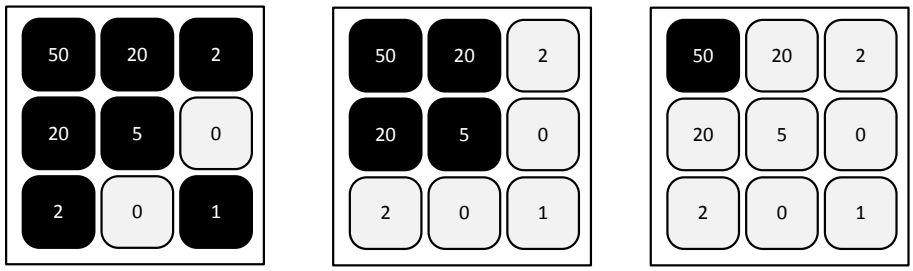


Figure 7.10: Three overlapped matrices indicating different fitting situations.

observed variants are allowed or 0 if none of the observed variants are allowed. In other words, the fitness is expressed as a value between 0 and 1 as it is in traditional process mining.

In order to make the approach robust in terms of noise, it is possible to take into consideration a threshold for the frequencies of variants. If the frequency of a variant is below the configured threshold, we consider that it is not observed in the log. Note that we can also set the threshold as a ratio to consider the relative frequency when deciding if a variant is observed.

Definition 7.6 (Fitness) *Let L be a sound XOC log, M be an OCBC model and P be a correlation pattern. Function $fitness_P \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ computes the fitness on the pattern level. We define the following notions:*

- $fitness_{P_1}(L,M,P) = \dfrac{|\{v \in V \mid freV(L,P,v) \geq 1\}|}{|\{v \in V_{CT} \mid freV(L,P,v) \geq 1\}|}$,
- $fitness_{P_2}(L,M,P) = \dfrac{|\{v \in V \mid freV(L,P,v) \geq \tau\}|}{|\{v \in V_{CT} \mid freV(L,P,v) \geq \tau\}|}$ *for some threshold* $\tau \in \mathbb{N}$,
- $fitness_{P_3}(L,M,P) = \dfrac{|\{v \in V \mid freV_{\%}(L,P,v) \geq \tau\}|}{|\{v \in V_{CT} \mid freV_{\%}(L,P,v) \geq \tau\}|}$ *for some threshold* $\tau \in [0,1]$, *and*
- $fitness_{P_4}(L,M,P) = \dfrac{freV(L,P,V)}{freV(L,P,V_{CT})}$,[1]

where $V = posV(M,P)$ is the set of allowed variants.

[1] In the remainder we assume that $\frac{x}{0} = 1$ unless indicated otherwise.

Definition 7.6 gives four solutions to compute fitness on the pattern level (i.e., corresponding to some pattern P). The first fitness notion ($fitnessP_1$) only counts the number of variants which are observed (i.e., $freV(L,P,v) \geq 1$) and allowed (i.e., $v \in V$ where V is the set of allowed variants), divided by the number of all observed variants ($\{v \in V_{CT} \mid fre(L,P,v) \geq 1\}$). This solution is sensitive to noise and is only recommended for logs without noise, which may not be applicable for real life logs.

The second solution can deal with noise by setting a threshold τ (i.e., an integer) to filter the infrequent variant. More precisely, a variant is observed if its frequency is equal to or larger than the given threshold (i.e., $freV(L,P,v) \geq \tau$). The third solution employs a similar idea, only changing the absolute frequency to a relative ratio. Unlike the second or the third solution which uses the frequency to identify if a variant is observed, the fourth solution computes fitness directly based on the frequency. The fitness is equal to the frequency of allowed variants (i.e., $freV(L,P,V)$) divided by the frequency of all variants in the variant matrix (i.e., $freV(L,P,V_{CT})$).

Solution	Threshold	fitness		
		Figure 7.10(a)	Figure 7.10(b)	Figure 7.10(c)
$fitnessP_1$	-	7/7(=1.0)	4/7(=0.57)	1/7(=0.14)
$fitnessP_2$	1	7/7(=1.0)	4/7(=0.57)	1/7(=0.14)
$fitnessP_2$	2	6/6(=1.0)	4/6(=0.67)	1/6(=0.17)
$fitnessP_2$	3	4/4(=1.0)	4/4(=1.0)	1/4(=0.25)
$fitnessP_2$	10	3/3(=1.0)	3/3(=1.0)	1/3(=0.33)
$fitnessP_2$	30	1/1(=1.0)	1/1(=1.0)	1/1(=1.0)
$fitnessP_3$	0.01	7/7(=1.0)	4/7(=0.57)	1/7(=0.14)
$fitnessP_3$	0.02	6/6(=1.0)	4/6(=0.67)	1/6(=0.17)
$fitnessP_3$	0.03	4/4(=1.0)	4/4(=1.0)	1/4(=0.25)
$fitnessP_3$	0.1	3/3(=1.0)	3/3(=1.0)	1/3(=0.33)
$fitnessP_3$	0.3	1/1(=1.0)	1/1(=1.0)	1/1(=1.0)
$fitnessP_4$	-	100/100(=1.0)	95/100(=0.95)	50/100(=0.5)

Table 7.2: The fitness derived by different solutions for each overlapped matrix in Figure 7.10.

Consider the three overlapped matrices in Figure 7.10 to understand how to compute fitness with different solutions. For the first overlapped matrix (i.e., Figure 7.10(a)), seven variants are observed (seven cells with numbers greater

than zero) and all these variants are allowed (cells colored in black). Therefore, the fitness derived based on $fitnessP_1$ is 7/7. The second matrix indicates that seven variants are observed, and four of them are allowed. Therefore, $fitnessP_1$ returns 4/7. Similarly, $fitnessP_1$ returns 1/7 for the third matrix, as shown in the first row in Table 7.2.

$fitnessP_2$ returns the same result when the threshold is set as 1. If we increase the threshold, the number of observed variants (whose frequencies are above the threshold) decreases. For instance, with a threshold 10 there are three observed variants, and all of them are allowed in the first and second matrix. Therefore, $fitnessP_2$ returns 3/3 as the fitness. In contrast, only one of the three observed variants is allowed in the third matrix, and $fitnessP_2$ returns 1/3, as shown in the fifth row in Table 7.2. According to a relative ratio, $fitnessP_3$ computes fitness in a similar way to $fitnessP_2$.

For these three matrices in Figure 7.10, the frequency of each variant is the same and the sum is 100. In the first matrix, all observed variants are allowed. In contrast, the allowed variants are observed 95 times and 50 times in the second matrix and third matrix, respectively. Therefore, $fitnessP_4$ returns 100/100, 95/100 and 50/100 for these three matrices.

7.3.4 Precision

While fitness evaluates whether the behavior in the log is possible with respect to the process model, precision evaluates how much behavior is allowed by the model which actually never happens in the log. A model having a poor precision is underfitting, i.e., it allows for behavior that is very different from what was seen in the event log. When the model becomes too general and allows for more behavior than necessary, it becomes less informative as it no longer describes the actual process. Therefore, precision is another important criterion, which should be taken into consideration when quantifying the conformance.

In this section, we refer to the idea of precision explained above and propose solutions to compute precision in the context of XOC logs and OCBC models. More precisely, we quantify the precision as the extent to which the allowed variants (by the OCBC model) are observed in the log based on the overlapped matrix derived in Section 7.3.2.

Figure 7.11 shows three overlapped matrices indicating situations with different precision. Intuitively, the first situation has perfect precision, since all behavior allowed by the model is observed in the log. The second overlapped matrix shows an almost precise situation, since most allowed behavior is observed. Differently, in the third matrix, two of the four allowed variants are not

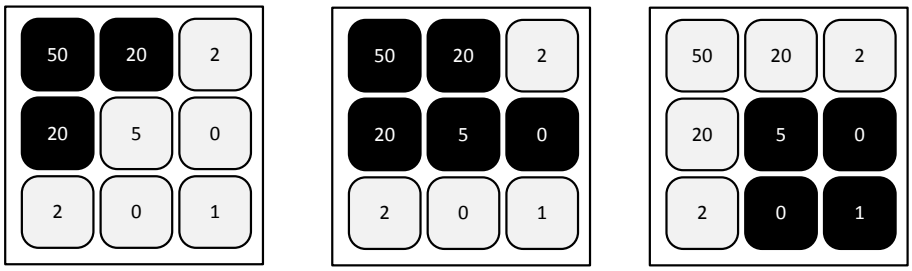


Figure 7.11: Three overlapped matrices indicating situations with different precision.

observed and the other two are observed infrequently, describing an imprecise situation.

After discussing precision in different situations, we explain our approach of computing precision. In general, the precision can be expressed as the ratio of the allowed and observed variants in all allowed variants. Based on this idea, the precision is 1 if all allowed variants are observed or 0 if none of the allowed variants are observed. Similar to the approach of computing fitness, we also take into consideration a threshold, i.e., if the frequency of a variant is below the configured threshold, we consider that the variant is not observed in the log.

Definition 7.7 (Precision) *Let L be a sound XOC log, M be an OCBC model and P be a correlation pattern. Function $precisionP \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ computes the precision on the pattern level. We define the following notions:*

- $precisionP_1(L,M,P) = \dfrac{|\{v \in V \mid freV(L,P,v) \geq 1\}|}{|V|}$,
- $precisionP_2(L,M,P) = \dfrac{|\{v \in V \mid freV(L,P,v) \geq \tau\}|}{|V|}$ *for some threshold* $\tau \in \mathbb{N}$,
- $precisionP_3(L,M,P) = \dfrac{|\{v \in V \mid freV_{\%}(L,P,v) \geq \tau\}|}{|V|}$ *for some threshold* $\tau \in [0,1]$, *and*
- $precisionP_4(L,M,P) = \dfrac{\sum_{v \in V} -p_v \log_2(p_v)}{\log_2(|V|)}$ *with* $p_v = \frac{freV(L,P,v)}{freV(L,P,V)}$,[2]

[2]This is based on the idea of entropy. As before, we assume that $\frac{x}{0} = 1$.

where $V = posV(M, P)$.

Definition 7.7 gives four solutions to compute precision on the pattern level (i.e., corresponding to some pattern P). The first precision notion ($precisionP_1$) only counts the number of variants which are observed (i.e., $freV(L, P, v) \geq 1$) and allowed (i.e., $v \in V$ where V is the set of allowed variants), divided by the number of all allowed variants (i.e., $|V|$, where $V = posV(M, P)$, defined in Definition 7.4). This solution is sensitive to noise and is only recommended for logs without noise, which may not be applicable for real life logs. The second and the third solutions deal with noise by setting a threshold τ, which uses the same filtering approach as used for computing fitness (cf. Definition 7.6).

Unlike the previous solutions which use the frequency to compute precision, the fourth solution computes precision based on the extent to which the allowed variants are evenly observed in terms of frequency. If all allowed variants have the same frequency (i.e., equally observed), the precision is 1, since $p_v = 1/|V|$ for each $v \in V$. If only one allowed variant v is observed (i.e., extremely unequally observed), the precision is 0, since $freV(L, P, V) = freV(L, P, v)$, i.e., $p_v = 1$.

Solution	Threshold	precision		
		Figure 7.11(a)	Figure 7.11(b)	Figure 7.11(c)
$precisionP_1$	-	3/3(=1.0)	4/5(=0.8)	2/4(=0.5)
$precisionP_2$	1	3/3(=1.0)	4/5(=0.8)	2/4(=0.5)
$precisionP_2$	3	3/3(=1.0)	4/5(=0.8)	1/4(=0.25)
$precisionP_2$	10	3/3(=1.0)	3/5(=0.6)	0/4(=0.0)
$precisionP_2$	30	1/3(=0.33)	1/5(=0.2)	0/4(=0.0)
$precisionP_3$	0.01	3/3(=1.0)	4/5(=0.8)	2/4(=0.5)
$precisionP_3$	0.03	3/3(=1.0)	4/5(=0.8)	1/4(=0.25)
$precisionP_3$	0.1	3/3(=1.0)	3/5(=0.6)	0/4(=0.0)
$precisionP_3$	0.3	1/3(=0.33)	1/5(=0.2)	0/4(=0.0)
$precisionP_4$	-	0.91	0.71	0.33

Table 7.3: The precision derived by different solutions for each overlapped matrix in Figure 7.11.

Consider the three overlapped matrices in Figure 7.11 to understand how to compute precision with different solutions. For the first overlapped matrix (i.e., Figure 7.11(a)), there are three allowed variants and all these variants are observed. Therefore, the precision derived based on $precisionP_1$ is 3/3. The

second matrix indicates that five variants are allowed, and four of them are observed. Therefore, $precisionP_1$ returns 4/5. Similarly, $precisionP_1$ returns 2/4 for the third matrix, as shown in the first row in Table 7.3. $precisionP_2$ returns the same result when the threshold is set as 1. If we increase the threshold, the number of observed variants (whose frequencies are above the threshold) decreases. For instance, with a threshold 3 the number of observed variants in the third matrix drops to 1. Therefore, $precisionP_2$ returns 1/4 for the third matrix. According to a relative ratio, $precisionP_3$ computes precision in a similar way to $precisionP_2$.

$precisionP_4$ computes precision based on the idea of entropy. Assuming that the frequency for each allowed variant in Figure 7.11(a) is 30 (without changing the sum, i.e., 90), the precision returned by $precisionP_4$ is 1 (indicating that allowed variants are equally observed). When changing the balance between allowed variants, the precision decreases. For instance, when the frequency is 50, 20 and 20 (real frequency in Figure 7.11(a)), the precision is 0.91.[3] If we make the allowed variants more imbalanced, the precision gets worse, e.g., $precisionP_4$ returns 0.71 and 0.33 for the second matrix and third matrix, respectively.

In traditional process mining, some approaches (e.g., [5]) also consider the non-fitting behavior when computing precision. Often the alignment technique is employed to find a fitting process instance (i.e., a complete activity sequence in the model that is most similar to the case) for each non-fitting case, and these alignments are used to measure the precision. By aligning cases, the information of fitting parts (of non-fitting cases) are also taken into consideration. In comparison, the pattern instances (corresponding to some correlation pattern) in our approach only have events of two activities, and they do not contain information of other patterns (unlike the cases which contain events of activities covering the whole process). It makes no sense to align the non-fitting behavior and take them into consideration. Therefore, in this chapter, we do not consider the non-fitting behavior (corresponding to variants which are not allowed by the model) when computing precision. More precisely, as shown in the four solutions, only allowed variants (i.e., $v \in V$) are considered and non-fitting parts are simply ignored.

7.3.5 Generalization

A process model should not restrict its allowed behavior to just the observed examples in a log, since the log may only cover a small part of all possible

[3] $\frac{-(50/90)\log_2(50/90)-(20/90)\log_2(20/90)-(20/90)\log_2(20/90)}{\log_2(3)} = 0.91$.

behavior in a process. In other words, a model should generalize enough to allow for other behavior (different from the sample behavior) from the same process. Generalization is a criterion related to this question. A model having a bad generalization is "overfitting". Overfitting is the problem that a very specific model is fitting a log, which only holds example behavior, too much and does not allow for other similar behavior.

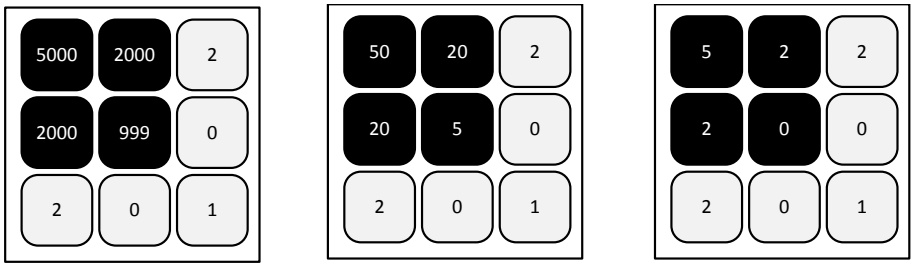
It is difficult to reason about generalization because this refers to unseen examples. In traditional process mining, generalization is described as the extent to which the model can explain new coming cases from the same process. Generalization is close to 1 if it is very likely that the next behavior from the process will fit the model. The generalization is close to 0 if it is very likely that new behavior from the process will violate the model. The idea is explained in more details next. We consider all points in the event log where event $e \in E$ is about to happen. Given an event e we can find all events e' that occur in the same state in model M. Every event can be seen as an observation of an activity in some state s. Suppose that state s is visited n times and that w is the number of different activities observed in this state. Suppose that n is very large and w is very small, then it is likely that a new event visiting this state will correspond to an activity seen before in this state. However, if n and w are of the same order of magnitude, then it is more likely that a new event visiting state s will correspond to an activity not seen before in this state.

In summary, the approach of computing generalization in traditional process mining is based on the number of times that a state is visited and the number of different activities observed in this state. This approach cannot apply to our task of computing generalization, since the pattern instances (corresponding to some correlation pattern) in our task only have events of two activities. In this case, for an event the number of different activities is 0 or 1, which can not generate a reasonable generalization based on the approach explained above. In this section, we consider generalization as a capability that the model (i.e., allowed variants) can explain new coming instances from the same process. Since the new coming instances are things in the future, computing generalization can be transformed into a question related to probability.

Let us consider some basic statistical notations to better understand generalization. Assume that we implement a trial which leads to exactly two mutually exclusive outcomes: "success" and "failure". In an experiment, we have n trials which succeed s times. The question is to predict the result of the next trial. We employ the Beta distribution to describe the probability distribution that

the next trial succeeds, denoted as X.[4] As we know nothing about the system before the n trials, therefore we assume that the prior distribution is the uniform distribution, i.e., $X = Beta(1,1)$. After the n trials which succeed s times, the posterior distribution is then $X = Beta(1+s, 1+n-s)$.

The example described above reveals the idea of computing generalization. Here we interpret "success" as "possible according to the model" and "failure" as "impossible according to the model". The pattern instances correspond to the trials, where n means the number of instances and s means the number of allowed instances. p is the assumed minimal probability that an instance is allowed, e.g., $p = 0.95$. Based on the discussion, generalization is considered as the possibility that the next instance will be allowed at a probability larger than p, i.e., $P(X > 0.95)$.



(a) good generalization (b) fair generalization (c) poor generalization

Figure 7.12: Three overlapped matrices indicating situations with different generalization.

Figure 7.12 shows three overlapped matrices indicating situations with different generalization. Intuitively, the first situation has very good generalization, since all the variants with high frequencies (i.e., 5000, 2000, 2000 and 999) are allowed. It is very likely that new coming instances from the same process correspond to the allowed variants, i.e., allowed by the model. In contrast, the third matrix shows a situation with poor generalization, since we have not observed frequent behavior corresponding to the allowed variants. In other words, it is very likely that new coming instances from the same process will violate the model.

[4] In Bayesian inference, the Beta distribution is the conjugate prior probability distribution for the Bernoulli, binomial, negative binomial and geometric distributions.

Definition 7.8 (Generalization) *Let L be a sound XOC log, M be an OCBC model and P be a correlation pattern. Function $generalizationP \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ computes the generalization on the pattern level such that $generalizationP(L, M, P) = P(X > p)$ where $V = posV(M, P)$, $p \in [0,1]$ and $X = Beta(1 + freV(L, P, V), 1 + freV(L, P, V_{CT} \setminus V))$.*

Definition 7.8 gives a solution to compute generalization on the pattern level. We assume that the likelihood that a coming instance is allowed by the model follows the Beta distribution. Based on the assumption, generalization is considered as the cumulative probability that the coming instance will be allowed at a probability larger than a given value $p \in [0,1]$, i.e., $P(X > p)$. We assume that the prior distribution is the uniform distribution, i.e., $X = Beta(1,1)$. Based on the log, the number of allowed instances are $freV(L, P, V)$ and the number of instances which are not allowed by the model are $freV(L, P, V_{CT} \setminus V)$, where $V = posV(M, P)$ is the set of allowed variants. Therefore, the posterior distribution is then $X = Beta(1 + freV(L, P, V), 1 + freV(L, P, V_{CT} \setminus V))$.

Table 7.4 shows the derived generalization in different situations, where n is the number of all instances and f is the number of instances which are not allowed by the model.[5] We assume $p = 0.95$, i.e., $generalizationP(L, M, P) = P(X > 0.95)$. Note that the derived values in the generalization column are reasonable in different situations, since they match the expected values according to the discussion in Figure 7.12. More precisely, if n is a small number (i.e., the first seven rows in Table 7.4), the derived generalization is low (corresponding to Figure 7.12(c)). Besides, when most instances are not allowed, the generalization is very low (i.e., the situation opposite to Figure 7.12(a)). Moreover, when n increases to a large number, the initial assumption has hardly any effect on the derived generalization. Therefore, the function in Definition 7.8 can precisely quantify generalization.

7.3.6 Lifting and Combining Criteria

Section 7.3.3, Section 7.3.4 and Section 7.3.5 give solutions to compute fitness, precision and generalization on the pattern level, respectively. In this section, we lift the criteria from the pattern level to the model level. More precisely, the criteria on the model level can be the average of all criteria on the pattern level, and it is possible to take into account frequencies.

[5] This applet (at http://homepage.divms.uiowa.edu/ mbognar/applets/beta.html) computes probabilities and percentiles for beta random variables.

n	f	$X = Beta(1+n-f, 1+f)$	generalization
0	0	$Beta(1,1)$	0.05
1	0	$Beta(2,1)$	0.0975
1	1	$Beta(1,2)$	0.0025
2	1	$Beta(2,2)$	0.00725
10	0	$Beta(11,1)$	0.4312
10	10	$Beta(1,11)$	0.0025
20	10	$Beta(11,11)$	0.0
100	0	$Beta(101,1)$	0.99438
100	100	$Beta(1,101)$	0.0
200	100	$Beta(101,101)$	0.0
100	1	$Beta(100,2)$	0.99408
100	5	$Beta(96,6)$	0.393
100	10	$Beta(91,11)$	0.01231
1000	1	$Beta(1000,2)$	1.0
1000	5	$Beta(996,6)$	1.0
1000	10	$Beta(991,11)$	1.0
1000	50	$Beta(951,51)$	0.46536

Table 7.4: The generalization derived in different situations by *generalizationP* where $p = 0.95$, $n = freV(L,P,V_{CT})$ and $f = freV(L,P,V_{CT} \setminus V)$.

Definition 7.9 (Lifting Criteria From Pattern Level To Model Level) *Let L be a sound XOC log and $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model. metricP represents a function to compute some criterion (fitness, precision or generalization) on the pattern level, i.e., fitnessP, precisionP or generalizationP. Function $metricM \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \rightarrow [0,1]$ computes some criterion on the model level. We define the following notions:*

- $metricM_1(L,M) = \frac{\sum_{P \in extP(ClaM,AOC)} metricP(L,M,P)}{|extP(ClaM,AOC)|}$,
- $metricM_2(L,M) = \frac{\sum_{P \in extP(ClaM,AOC)} metricP(L,M,P) * freV(L,P,V_{CT})}{\sum_{P \in extP(ClaM,AOC)} freV(L,P,V_{CT})}$.

where $V = posV(M,P)$.

Function $metricM_1$ lifts the criteria from the pattern level to the model level by computing the average of the criteria of all patterns. The criteria for different

patterns may have different weights, i.e., if a pattern has more corresponding instances, its criteria should be considered more in the criteria on the model level. Therefore, $metricM_2$ uses the frequency corresponding to each pattern as its weight, and compute the average.

Fitness, precision or generalization quantifies the conformance on one specific perspective. It is possible to combine these three criteria as one score to comprehensively evaluate the conformance from multiple perspectives (like the F1 Score which combines the precision and recall in data mining).

Definition 7.10 (Combined Criteria) *Let L be a sound XOC log, M be an OCBC model and P be a correlation pattern. Function $scoreP \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ returns a score which combines fitness, precision and generalization on the pattern level, such that $scoreP(L,M,P) = (w_f * fitnessP(L,M,P) + w_p * precisionP(L,M,P) + w_g * generalizationP(L,M,P))/(w_f + w_p + w_g)$ where*

- *w_f, w_p, and w_g $(0 \leq w_f, w_p, w_g \leq 1)$ are weights, and*
- *fitnessP, precisionP and generalizationP are functions to compute fitness, precision and generalization on the pattern level, respectively.*

The score is impacted by the weights configured by users. For instance, if one cares more about the fitness perspective when quantifying the conformance, the fitness weight can be set as a larger value than weights of precision and generalization. Without preference, all weights can be set as 1. In this way, users can interact with the approach when quantifying conformance.

7.3.7 Customization of Behavioral Constraints by Criteria

Chapter 6 introduced approaches to discover OCBC models. On the behavioral perspective, the basic idea of these approaches is to discover constraints which are satisfied by the log. If we evaluate the discovery approaches based on the three criteria defined in previous sections, it is obvious that these approaches mainly focus on the fitness dimension. Therefore, it is possible that the discovered models suffer bad precision and generalization.

In some situations, the precision and generalization are also important for models and often there is a trade-off between these criteria. For instance, a model with a perfect precision tends to be overfitting and suffer a bad generalization. It turns out to be challenging to balance these criteria and the trade-off should match the users' requirements. For instance, based on some domain knowledge (a priori knowledge), one may want to derive a model with perfect fitness, good

precision and arbitrary generalization. Therefore, it is helpful to enable users to guide the discovery task. [18] proposed an approach named Evolutionary Tree Miner (ETM) algorithm, which can steer the discovery process based on user-defined weights for different quality dimensions.

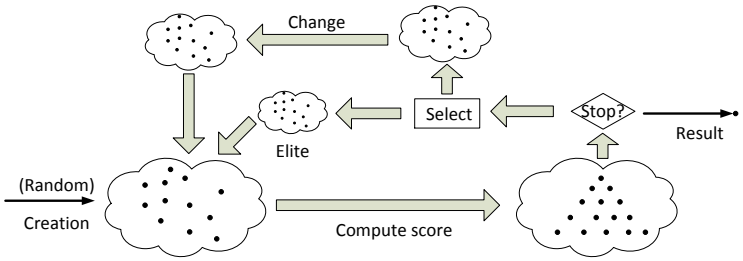


Figure 7.13: The idea for the Evolutionary Tree Miner (ETM) algorithm.

As shown in Figure 7.13, the ETM algorithm uses the genetic manner to discover models with high scores cyclically. The input of the algorithm is an event log describing observed behavior. In the initial step a population of random process trees is generated where each activity occurs exactly once in each tree. The quality dimensions are calculated for each candidate in the population. Using the weight given to each dimension the overall score of the process tree is calculated. In the next step, certain stop criteria are tested such as finding a tree with the desired overall score. If none of the stop criteria is satisfied, the candidates in the population are changed and the score is again calculated. This is continued until at least one stop criterion is satisfied and the best candidate with the highest score is then returned.

Triggered by the idea explained above, we also allow the user to specify the relative importance of each dimension beforehand (i.e., configure weights for fitness, precision and generalization) and search the best candidate with highest score. Differently, as shown in Figure 7.14, the input for our approach is a set of pattern instances and each candidate is an overlapped matrix rather than a process tree. More precisely, since each variant can be colored in black or not, (i.e., two possibilities), we have 2^9 possible different colored matrices. Each colored matrix is combined with the frequency matrix (derived based on the pattern instances), resulting in an overlapped matrix. Based on the configured weights for fitness, precision and generalization, we compute the score for each overlapping matrix. After 2^9 loops, the scores for all overlapped matrices are

derived. We select the best one(s) with the highest score(s) and then transform the corresponding colored matrix into a set of constraint types.

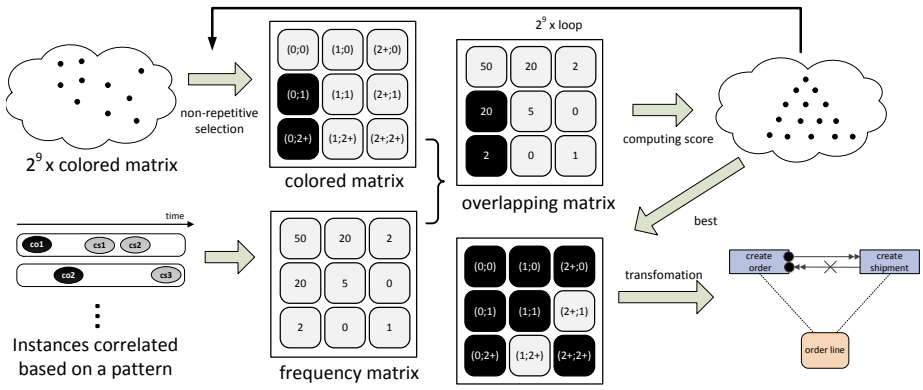


Figure 7.14: The idea for discovering customized constraints.

Definition 7.11 (Computing Scores based on Configured Weights) *Let L be a sound XOC log and P be a correlation pattern. Function $scoreV \in \mathcal{U}_L \times \mathbb{P}(V_{CT}) \times \mathcal{U}_P \rightarrow [0,1]$ returns a score which combines fitness, precision and generalization on the pattern level, such that $scoreV(L,V,P) = (w_f * fitnessV(L,V,P) + w_p * precisionV(L,V,P) + w_g * generalizationV(L,V,P))/(w_f + w_p + w_g)$ where*

- *$V = posV(M,P)$ corresponds to a colored matrix,*
- *w_f, w_p, and w_g $(0 \leq w_f, w_p, w_g \leq 1)$ are configured weights, and*
- *fitnessV, precisionV and generalizationV are functions corresponding to fitnessP, precisionP and generalizationP, respectively.*

In Definition 7.11, *fitnessV*, *precisionV* and *generalizationV* are functions to compute criteria for a colored matrix V (i.e., any $v \in V$ is colored in black while others are not). They have the same expressions with *fitnessP*, *precisionP* and *generalizationP*, respectively, but take a different parameter as input (i.e., V rather than M). For instance, $fitnessP_1(L,M,P) = \frac{|\{v \in V \mid freV(L,P,v) \geq 1\}|}{|\{v \in V_{CT} \mid fre(L,P,v) \geq 1\}|}$ where $V = posV(M,P)$, and $fitnessV_1(L,V,P) = \frac{|\{v \in V \mid freV(L,P,v) \geq 1\}|}{|\{v \in V_{CT} \mid fre(L,P,v) \geq 1\}|}$. The reason that

we change the parameter is that the model does not exist yet at the stage of computing scores. Note that the parameter M is not involved in all expressions and it is only used to compute V (i.e., $V = posV(M, P)$). Therefore, the expression can still return the same result, although the parameter changes from M to V.

By correlating events in a log L based on a correlation pattern P, we derive a set of pattern instances. Function *freV* is employed to map these instances onto the variant matrix, resulting in a frequency matrix. Given a colored matrix V, an overlapped matrix is generated by overlapping V onto the frequency matrix. Definition 7.11 gives an approach to compute the score of the overlapped matrix by combining fitness, precision and generalization on the pattern level with configured weights. The score is impacted by the weights configured by users. For instance, if one cares more about the fitness, the fitness weight is configured as a number larger than weights of precision and generalization. If one wants to discover a balanced model at these three perspectives, all weights are configured as the same number. In this way, users can interact with the discovery approach to derive customized models.

Definition 7.12 (Searching for the Best Colored Matrix) *Let L be a sound XOC log and P be a correlation pattern. Function $besMatrix \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathbb{P}(V_{CT})$ returns a colored matrix $V_{bes} = besMatrix(L, P)$ with the best score, i.e., $scoreV(L, V_{bes}, P) = \max_{\forall V \in \mathbb{P}(V_{CT})} scoreV(L, V, P)$.*

Definition 7.12 searches the colored matrix with the best score by exploring all possible (i.e., 2^9) colored matrices. Note that we do not use the genetic idea since the searching space is not big. Assuming that the variant matrix is not divided into 9 cells, but a lot more cells, the genetic manner could be used to reduce the exploration space.

Definition 7.13 (Discovery of Customized Behavioral Constraints) *Let L be a sound XOC event log, P be a correlation pattern and CT be a set of constraint types. Function $disCC \in \mathcal{U}_L \times \mathcal{U}_P \times \mathbb{P}(CT) \rightarrow \mathbb{P}(\mathcal{U}_{Con})$ discovers customized behavioral constraints, such that $disCC(L, P, CT) = \{con \mid con = idCon(P, ct) \wedge ct \in CT_{bes} \setminus CT_{red}\}$ where*

- *$CT_{bes} = \{ct \in CT \mid \forall v \in V_{bes} : v \subseteq ct\}$, where $V_{bes} \subseteq V_{CT}$ is the best colored matrix, and*
- *$CT_{red} = \{ct \in CT_{bes} \mid \exists ct' \in CT_{bes} : ct' \subsetneq ct\}$, which is the set of redundant constraint types.*

Based on the best colored matrix, Definition 7.13 discovers customized behavioral constraints. More precisely, if all black (i.e., allowed) variants in the best colored matrix consist with a constraint type ct (i.e., $\forall v \in V_{bes} : v \subseteq ct$), a constraint of this type is discovered, i.e., $con = idCon(P, ct)$. Note that it is possible that there exist redundant constraint types. Here, we employ the same approach as used to discover constraints based on frequent variants (cf. Chapter 6) to filter the redundant constraints.

In this section, based on the defined criteria we propose an approach, which allows users to seamlessly steer the discovery process based on preferences with respect to three dimensions, i.e., fitness, precision and generalization.

7.4 Evaluation

In this section, the OCBC conformance checking approach is evaluated. More precisely, we first set up an experimental environment to derive XOC logs, and inject some known deviations in these logs. Based on the generated logs and reference OCBC models, we verify if the approach can detect these injected deviations. At last, we compare our approach with other conformance checking techniques.

7.4.1 Experimental Design

The experimental environment is designed in a controlled manner. Figure 7.15 shows the details about how to evaluate the OCBC conformance checking technique. In general, starting from a particular business scenario, the upper branch simulates the scenario to generate an XOC log (representing observed behavior), the lower branch designs a model to describe the scenario (representing allowed behavior), and, at last, the log and the model are checked to diagnose the problems.

The conformance checking approach highly depends on the availability of XOC event logs following the event log notion defined in Chapter 4. Such event logs are different from standard XES, MXML, and CSV log files in two respects: (i) there is no single case notion, and (ii) each event is related to an object model describing the "state" of the process. This aligns well with the way that actual information systems work: data are stored in a database and transaction update the database. Here we use *Dolibarr ERP/CRM* to illustrate the feasibility of the approach and the availability of the data assumed.

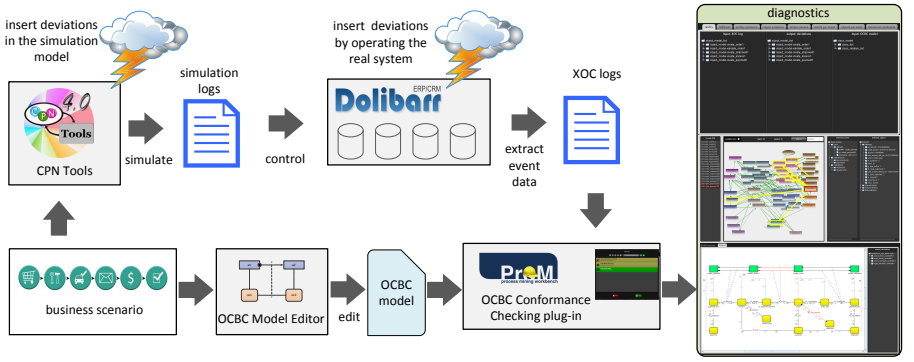


Figure 7.15: The approach to evaluate the OCBC conformance checking approach and tooling.

Given a process scenario, we design a simulation model with CPN Tools (`cpntools.org`). (See [72, 144] for an introduction to modeling using Colored Petri Nets (CPNs) and the CPN Tools environment.) By simulating complex process involving multiple interacting entities in the simulation model with CPN Tools, a simulation log is generated. We interpret the simulation log to automatically operate the Dolibarr ERP/CRM and populate the corresponding database.[6] For example, if an order is created in the simulation and exported in the simulation log, it is also created in the real system. By running the simulation, the tables of Dolibarr get filled with information about orders, customers, deliveries, etc. After this, we extract XOC event logs from the database of Dolibarr. In order to conduct controlled experiments, we inject some deviations into the normal log and check whether the deviations that are injected can actually be discovered. Based on the normative model, we can add deviating behavior into normal behavior by two methods:

- *Adding deviating paths into the simulation model.* This method is used to add deviations on the model level. More precisely, we model the deviating behavior as a path violating the normative model. For instance, we can add an alternative path which skips the activity "create shipment" when some attributes of orders satisfy predefined rules. This method generates normal behavior and deviating behavior at the same time when simulating

[6] More information is at http://www.win.tue.nl/ocbc/softwares/data_generation.html.

the model, which allows generating large numbers of deviations.

- *Adding deviating behavior manually on the real system.* This method is used to add deviations on the instance level, i.e., operating the real system deliberately violating the normative scenario. For instance, after creating an order, we never create shipments for this order. This method makes it possible to intertwine the simulated data with real behavior, i.e., we create normal behavior by simulating the model and inserting deviations manually on the real system.

The lower branch in Figure 7.15 shows how to derive OCBC models. To illustrate the approach, let us focus on the *order-to-cash* scenario in Dolibarr. Based on the scenario, we create a normative OCBC model with *OCBC Model Editor* in ProM, as shown in Figure 7.16. The model indicates that there are eight classes and four activities involved in this process. The class relationships reveal the constraints between classes, e.g., each order line should eventually have a corresponding shipment line indicated by $r9$ (this is consistent with the real scenario where each order line is eventually shipped to the corresponding customer). The seven behavioral constraints (i.e., $con1 \sim con7$) present restrictions assigned on the temporal order between events of different activities. For instance, $con6$ indicates that each "create order" event is followed by one or more corresponding "create shipment" events while $con7$ requires each "create shipment" event is preceded by precisely one corresponding "create order" event (since Dolibarr does not enable creating shipments covering multiple orders). The eight AOC relations (i.e., ① ~ ⑧) specify the cardinaltiy constraints between activities and classes. For example, ⑤ shows a one-to-one correspondence between "create order" events and "order" objects, i.e., if an "order" object is observed, the corresponding "create order" activity needs to be executed once and vice versa.

7.4.2 Detecting Deviations

In this section, we manually inject typical deviations (which may really happen in daily transactions), and detect them using our approach. Figure 7.17 shows the object-centric event data (cf. Chapter 3), generated by simulating Dolibarr. Note that we scope the data in a consistent way with the reference model in Figure 7.16, i.e., the events and objects only contain instances of the activities and classes in the reference model. For simplicity, we omit the objects of the "customer" class, since they are not referred to by any events.

As the OTC process contains the behavioral perspective, the data perspective

and the interactions between the two perspectives, we inject deviations of three categories accordingly. These deviations are highlighted in Figure 7.17 and explained as follows:

- *Deviations on the behavioral perspective.* Like other modeling languages such as Petri nets, OCBC models support checking conformance on the behavioral perspective. In the normal scenario, each "create invoice" event is followed by at least one "create payment" event, indicated by the constraint $con2$ in Figure 7.16. In the experiment, we add a deviation violating $con2$, i.e., a "create invoice" event ("ci204" in Figure 7.17) is never followed by corresponding "create payment" events.
- *Deviations on the data perspective.* A challenge for detecting behavioral deviations is how to detect implicit deviations. For instance, a "create order" event has corresponding "create shipment" events but does not have sufficient ones, i.e., order lines created by the "create order" event are not totally shipped to the corresponding customer. This implicit deviation is indeed violating our scenario but satisfying the behavioral constraint (i.e., $con6$, which requires a one-to-many relation between "create order" events and "create shipment" events). As OCBC models have a data perspective, it is possible to transform such implicit behavioral deviations onto the data perspective. For example, the deviation mentioned above can be interpreted as "some order lines have no corresponding shipment lines", and be detected by checking the cardinality constraints on the

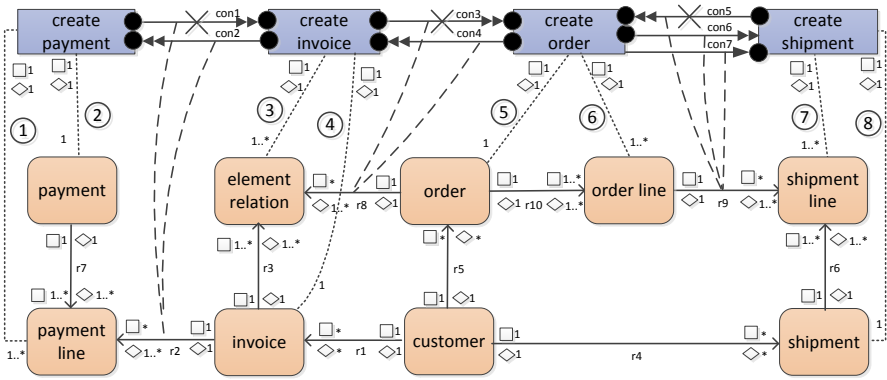


Figure 7.16: The normative OCBC model of the *order-to-cash* process in Dolibarr.

class relationship $r9$. In the experiment, we add a deviating "order line" object $order\_line734$ ("ol734" in Figure 7.17) which has no corresponding "shipment line" objects.

- *Deviations related to the interactions between two perspectives.* In the Dolibarr system, when an invoice is created, it is generally linked to one or more existing orders. As a result, one or more element relations (showing the correspondence between the invoice and the orders) are created when a "create invoice" event happens, indicated by the cardinality "1..*" of the AOC relation ③. In reality, a possible deviating situation is that one forgets to link one invoice to any orders. In our experiment, we mimic this situation, i.e., create an invoice ("ci205" in Figure 7.17) without any element relations, resulting in a deviation violating the constraints (i.e., "1..*" of ③) on the interactions of two perspectives.

Note that the deviations of different types depend on and impact each other. For instance, the deviation (related to the interactions) that the "create invoice" event $ci205$ does not create an "element relation" object leads to two other additional deviations: object "i205" has no corresponding "element relation"

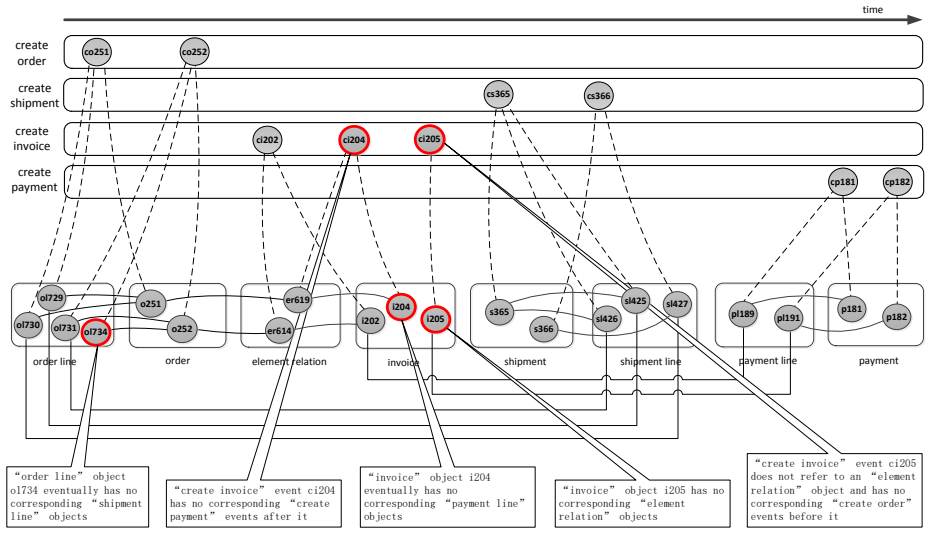


Figure 7.17: An illustration of the injected deviations in the normal log generated in the order-to-cash process in Dolibarr.

objects (a deviation on the behavioral perspective) and event “ci205” cannot be correlated to a “create order” event before it (a deviation on the behavioral perspective).

After loading the XOC event log with deviations (extracted from the data shown in Figure 7.17) and the OCBC model (designed based on the scenario) into ProM, we check the conformance to see if the inserted deviations can be identified. In our experiments, the injected deviations mentioned above can be totally detected and displayed clearly in three views.

As we can see in Figure 7.18, the type view reports all detected deviations which are grouped by types. If there is a star ($*$) symbol attached after the name of a type, it indicates that there exist deviations of this type. Figure 7.18 shows the diagnosis result corresponding to the “fulfillment” type. The left panel shows the object models from the log and the right panel presents the class model from the OCBC model, which are taken as input for conformance checking. The middle panel displays the detected deviations, e.g., the deviating object “order_line734”.

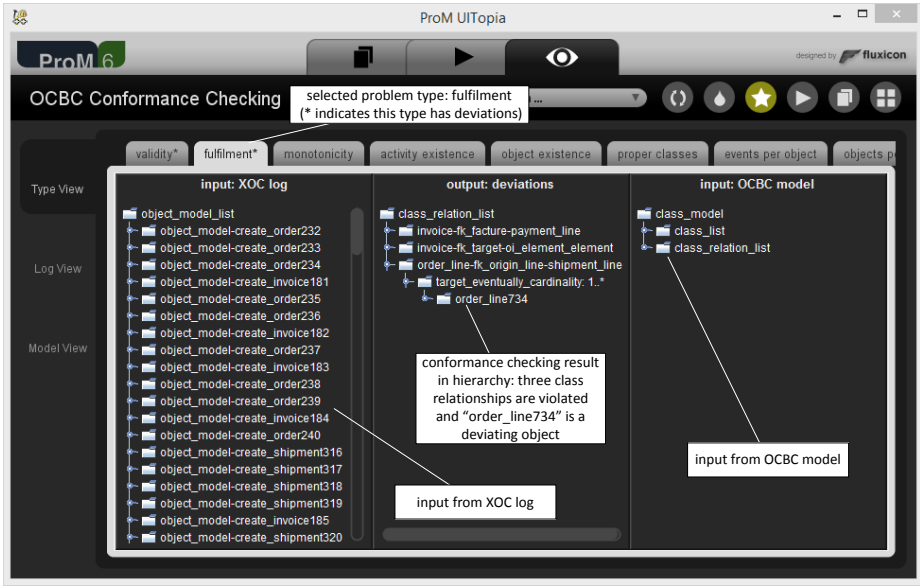


Figure 7.18: The conformance checking result in the type view.

Figure 7.19 shows the diagnosis result using a helicopter view (i.e., model view), by highlighting the violated constraints and cardinalities in the model. In summary, there are six constraints violated and highlighted in red, which corresponds to the injected deviations described in Figure 7.17. For instance, the annotation (1) indicates that some “create invoice” events are deviating since they are not followed by “create payment” events, and it corresponds to the deviating event $ci204$. After obtaining a general idea about the deviations on a high level, we can use the log view in Figure 7.20 to see the deviating instances.

Consider for example the deviation (in Figure 7.17) that object “order_line734” eventually has no corresponding “shipment_line” objects to understand how the three views display deviations from different angles. In Figure 7.19, the model view indicates the target cardinality (i.e., $\Diamond 1..^*$) of the class relationship between “order_line” and “shipment_line” is violated. By clicking this relationship, the “output:deviations” panel on the right displays the deviating object

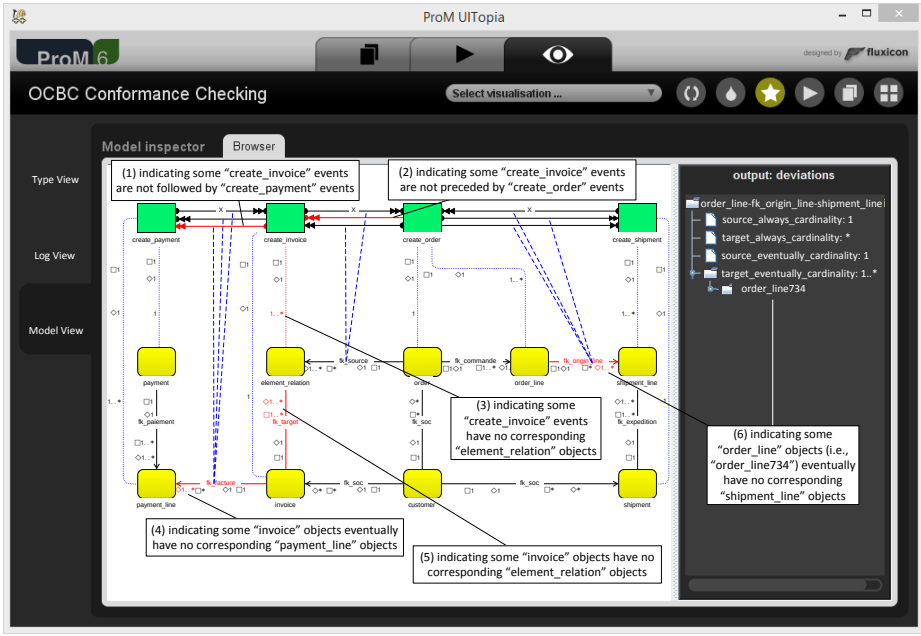


Figure 7.19: The conformance checking result in the model view.

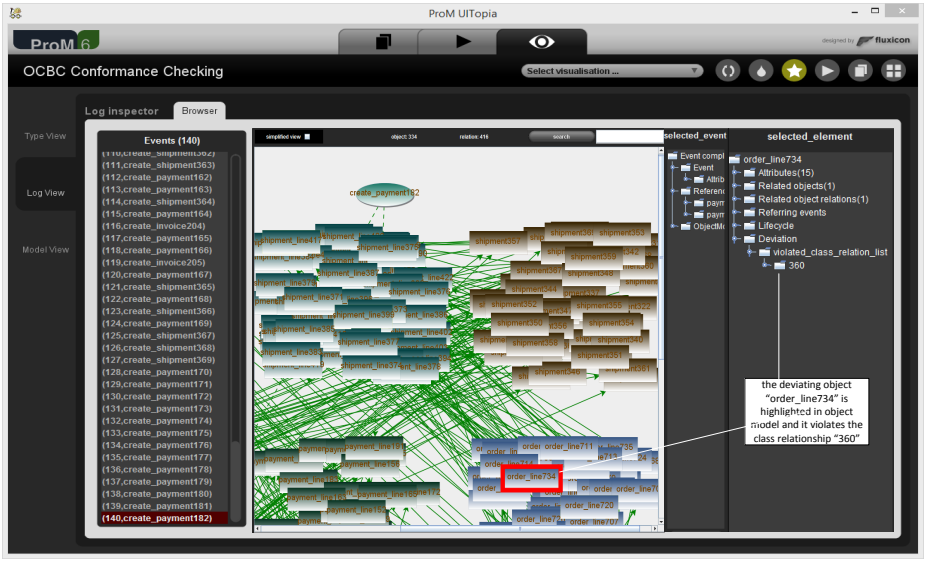


Figure 7.20: The conformance checking result in the log view.

"order_line734" under its corresponding cardinality. As "order_line734" violates the "eventually" cardinality, it is only considered as a deviation at the end of the log. As shown in Figure 7.20, "order_line734" is highlighted in the object model (in the middle panel) corresponding to the last event and the violated constraint id (i.e., "360") is displayed under the "Deviation" branch in the right panel. In the type view, this deviation is classified into the "fulfillment" type.

7.4.3 Comparison

In Section 7.4.2, we applied the OCBC conformance checking technique to the data generated by Dolibarr. The diagnosis result shows that all inserted deviations can be found and presented through three views. In this part, we apply the traditional technique to the same data and compare its results with the derived deviations using our approach.

The replay and alignment techniques often take an XES log and a Petri net as input. First, we derive the XES log based on the object-centric event data (cf. Chapter 3) shown in Figure 7.17. As the data are from artifact-centric

information system, there is no case notions to correlate events. Therefore, we correlate events based on objects, e.g., two events are correlated if they refer to the same object or two connected objects (cf. Chapter 6). Figure 7.21 shows the correlated events. For instance, event $co251$ is correlated to $ci204$, as $co251$ refers to the object $o251$, $ci204$ refers to the object $er619$ and $o251$ is connected to $er619$, as shown in Figure 7.17.

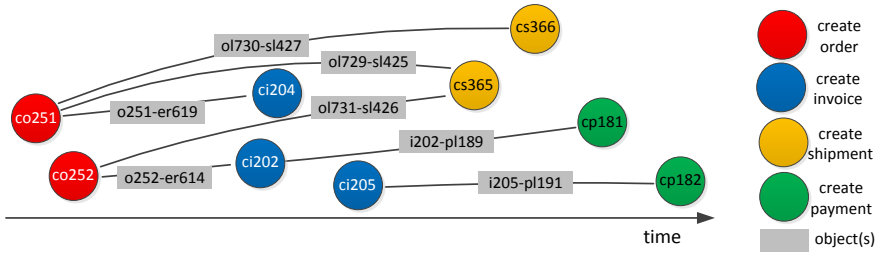


Figure 7.21: Correlating events based on objects.

Then, we transform the correlated events in Figure 7.21 into XES logs. Different from XOC logs, XES logs assume a case notion to integrate events into process instances. Here, we consider "order" as the case notion and derive an XES log with two cases in Figure 7.22, as there exist two "order" objects $o251$ and $o252$. The second case includes events $co252$, $ci202$, $cs365$ and $cp181$, because these events are linked to $o252$ (directly or indirectly). Note that the events $ci205$ and $cp182$ are discarded when generating the XES log, because they are not linked to any "order" objects.

Figure 7.22: The generated XES log assuming that "order" is the case notion.

According to the OTC scenario, we design a Petri net to describe the business process as shown in Figure 7.23. More precisely, after an order is created we

have two branches. The top branch shows that multiple invoices can be created for the order and there exists a many-to-many relationship between invoices and payments, i.e., one invoice can be paid multiple times and one payment can cover multiple invoices. The two implicit transitions (i.e., t) describe this relationship. Independent from the top branch, the bottom one indicates that multiple shipments can be created to deliver order lines in the order. After all shipments and payments, the process ends.

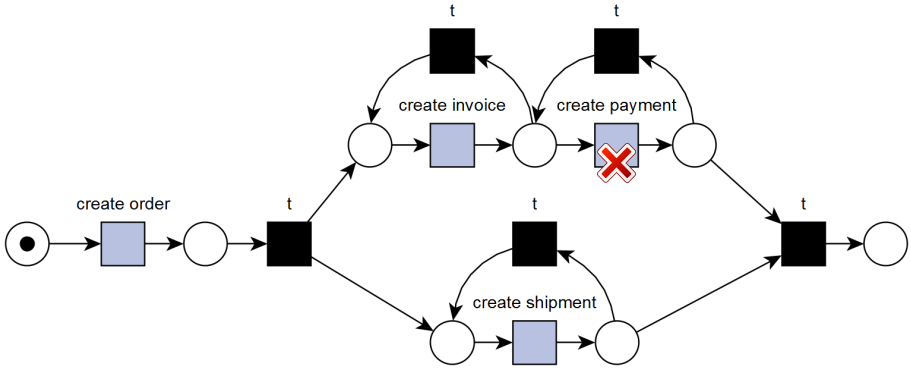


Figure 7.23: A Petri net to describe the OTC business process, including conformance checking result with respect to the log in Figure 7.22.

Using existing techniques to check the conformance between the log in Figure 7.22 and the model in Figure 7.23, it is possible to detect some deviations on the behavioral perspective. For instance, the deviation that "create invoice" event $ci204$ is not followed by a "create payment" event is detected, indicated by the cross symbol. However, the deviation that the "create invoice" event $ci205$ has no corresponding "create order" event before it is not detected, as $ci205$ is not included in the XES log in Figure 7.22.

The conformance checking on the whole process with an assumed case notion of "order" fails to detect all deviations, as a case notion only provides a view of the process from a particular angle. It is possible to detect the deviating "create invoice" event $ci205$ by considering the sub-process related to "invoice". Figure 7.24 shows the derived XES log based on the "invoice" case notion, and Figure 7.25 describes the sub-process related to "invoice". Note that the "create shipment" activity is discarded in the sub-process because it is not related to "invoice". After conformance checking, the deviating event "create invoice" event

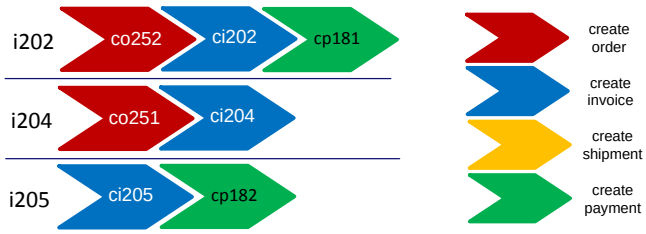


Figure 7.24: The generated XES log assuming that "invoice" is the case notion.

$ci205$ is detected, as $ci205$ has no "create order" event before it, indicated in the third case in Figure 7.24.

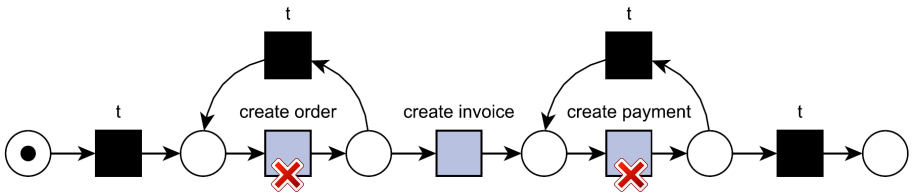


Figure 7.25: A Petri net to describe the sub-process related to "invoice" (i.e., the life-cycle of an invoice), including conformance checking result with respect to the log in Figure 7.24.

In some situations, it is possible to detect the deviations on the data perspective. Consider for example the deviating object $ol734$ (in Figure 7.17), which has no corresponding "shipment line" objects. By considering "order line" as an artifact, we extract a corresponding XES log with four cases in Figure 7.26, and use the model in Figure 7.27 to describe its life-cycle. After conformance checking, the result shows that the event $co252$ has no following "create shipment" event in the case corresponding to $ol734$, indicating that $ol734$ is deviating.

According to the artifact-centric approach illustrated above, the deviations on the data perspective can be detected if we can somehow transform them into deviations on the behavioral perspective, e.g., the deviating object $ol734$. However, if the deviation is related to objects which are not referred to by events, it is impossible to transform them into deviating events. For instance, assume that an "order" object does not have a corresponding "customer" object,

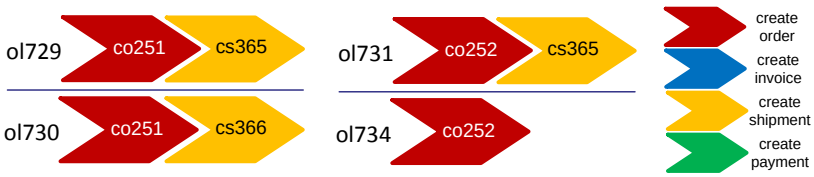


Figure 7.26: The generated XES log for the “order line” artifact.

which violates the class relationship $r5$ in Figure 7.16. This deviation cannot be detected by the approach, because “customer” objects are not referred to by any events.

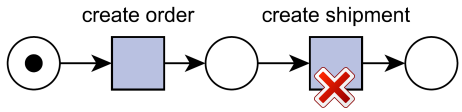


Figure 7.27: A Petri net to describe the life-cycle of the “order line” artifact.

In summary, traditional conformance checking techniques support detecting deviations on the behavioral perspective. They fail to efficiently detect deviations related to multiple instances and the data perspective, although it is possible to overcome some of the limitations by using some expertise such as choosing proper case notions, sub-processes or artifacts. In comparison, the OCBC conformance checking technique is more powerful in this situation. It can detect the deviations related to the data perspective and interactions in a straight-forward manner. Besides, the implicit deviations on the behavioral perspective can be detected when taking into consideration the data perspective.

7.5 Related Work

Conformance checking compares an existing process model with an event log of the same process. The comparison shows where the real process deviates from the modeled process. Moreover, it is possible to quantify the level of conformance based on the criteria such as fitness, precision and generalization. In this section, we discuss the literature related to approaches to check the conformance between a log and a model, and criteria to quantify the conformance.

Token-based replay [35, 123, 125] uses both an event log and a process model as input, i.e., history is replayed on the model to analyze various phenomena. More precisely, the numbers of produced, consumed, missing and remaining tokens are counted while replaying the event log. For instance, if an activity in the event log is not enabled, then a missing token is added. These numbers can be used to diagnose conformance problems. A typical application is to compute fitness based on the numbers. After replaying a trace σ in a log on top of a model, the fitness of a trace σ is computed based on four counters: p (produced tokens), c (consumed tokens), m (missing tokens), and r (remaining tokens). The fitness is then calculated as $\mathit{fitness}(\sigma, N) = \frac{1}{2}(1 - \frac{m}{c}) + \frac{1}{2}(1 - \frac{r}{p})$. The same approach can be used to analyze the fitness of a log consisting of many cases: simply taking the sums of all produced, consumed, missing, and remaining tokens, and applying the same formula.

Token-based replay can differentiate between fitting and non-fitting cases. It is easy to understand and can be implemented efficiently. However, this approach has some drawbacks. Intuitively, fitness values tend to be too high for extremely problematic event logs. If there are many deviations, the Petri net gets "flooded with tokens" and subsequently allows for any behavior. The approach is also Petri-net specific and can only be applied to other representations after conversion.

Alignments are introduced to overcome these limitations [3, 5–7] . Given a trace in the event log, the closest path in the model is computed by solving an optimization problem. In alignment approaches, model moves and log moves are used to describe the misalignment, and they are easier to interpret than missing and remaining tokens. By assigning a cost function to this misalignment, fitness is computed based on the total costs.

The replay and alignment techniques mainly focus on fitness. There exist techniques which can compute other criteria such as precision and generalization. In [104] a prefix automaton is constructed based on the event log to count so-called escaping edges. By quantifying these edges and their frequency, it provides an accurate measurement of the precision dimension. [105] extends and complements the technique introduced in [104], by additionally considering potential variability in the event log. Moreover, it introduces a novel technique to estimate the confidence interval of the metric, which estimates the robustness of the precision. Note that traces in the log do not need to be completely fitting. In [104, 105], the non-fitting parts are simply ignored, i.e., only a fraction of the event log can be used for computing precision, resulting in unreliable precision measurements. In [4], by aligning the event log and the model, the pre-

alignment makes it possible to measure precision more accurately, even in case of deviations. [137] shows how to compute the generalization criterion, which measures if the model is "overfitting" (i.e., the model explains the particular sample log, but it is unlikely that another sample log of the same process can be explained well by the current model).

Besides checking conformance on Petri nets, [43, 44] check the conformance of artifact-centric models expressed in terms of proclets. These papers show that process instances *cannot* be considered in isolation as instances in artifact-centric processes may overlap and interact with each other. This complicates conformance checking but the problem can be decomposed into a set of smaller problems, that can be analyzed using conventional conformance checking techniques. These artifact-centric conformance checking techniques do not relate control-flow to some overall data model (like the class model in OCBC models).

Also related is the work on conformance checking of *declarative models* [31, 32]. [31] proposes an approach to compute the fitness of Declare models. It creates a constraint automata for each constraint to derive the optimal alignment. It also presents some techniques to remove the search space. [32] discusses how to compute fitness, precision and generalization for Declare models. However, this work does not consider multiple intertwined instance notions and also does not relate control-flow to some overall data model.

These techniques and criteria explained above only cover the control-flow perspective. There also exist approaches which check conformance on other perspectives. [165] constructs a timed business process model describing activities from mobile services, on the basis of the specification and verification of temporal constraints. Then conformance checking is implemented by verifying how compliant the constructed model and a new arrival event log. [59] extends BPMN to support modeling Cloud resources and their pricing strategies. Then the matching between temporal constraints of Cloud resources and the temporal constraints of BP activities are checked. These two approaches check the conformance on the control-flow and time perspectives in terms of temporal constraints. [34] describes an approach that aligns event log and model, and takes data and resources into account when checking process conformance. Multi-perspective conformance checking [97] considers the conformance of process to multiple perspectives of a process model (i.e., control-flow, data, resources, time) at the same time.

7.6 Summary

In this chapter, we proposed techniques to check conformance based on OCBC models in terms of the behavioral perspective, data perspective and the interactions in between. In this way, we overcome the problems of existing approaches that instances are considered in isolation and constraints on the data perspective are not taken into account. Hence, we can now detect and diagnose a range of conformance problems that would have remained undetected using conventional process model notations.

The conformance checking task is split into two parts in this chapter. The first part is diagnosing the local conformance. We abstract a set of rules representing an OCBC model and detect nine types of conformance problems. The second task is to define three criteria, i.e., fitness, precision and generalization to quantify the conformance on the global level. Based on the criteria, a model discovery algorithm is proposed to discover behavioral constraints satisfying the configured criteria. At last, we evaluate our conformance checking approach based on the data generated in a real system Dolibarr by simulating the system.

In this chapter, the conformance diagnosis reveals the deviations about the instances on the entities level, i.e., events and objects. In the future, it is possible to check conformance on the attribute level, by taking into consideration the attribute values.

Chapter 8

OCBC Performance Analysis

In Chapter 6, we discovered OCBC models from XOC logs to reveal the real executions of business processes. In Chapter 7, we compared XOC logs with OCBC models to check conformance and detect deviations (which suggest fraud or inefficiencies). These two chapters cover two types of process mining, i.e., discovery and conformance. We now shift our attention to another type of process mining: enhancing process models with performance-related information. More precisely, by projecting XOC logs onto OCBC models, timestamps and frequencies of activities can be used to identify bottlenecks and diagnose other performance related problems.

This chapter is organized as follows. In Section 8.1, we motivate the performance analysis based on OCBC models by discussing the challenges not addressed by conventional performance analysis approaches. When these approaches are applied to artifact-centric information systems, serious problems emerge. In Section 8.2, we analyze the performance on the time perspective, by presenting the performance with dotted charts, column charts and indicators. Section 8.3 analyzes the performance on the frequency perspective, highlighting the "high way" (i.e., frequent parts) of the process. In Section 8.4, we evaluate our performance analysis approach and compare it with other related techniques. Section 8.5 reviews the related work while Section 8.6 concludes this chapter.

8.1 Motivation for OCBC Performance Analysis

Business processes form the heart of companies, no matter if they are small or large. Competitiveness of companies depends on continuous improvement of business processes, because "every good process eventually becomes a bad process" without improvement in a fast-changing era [61]. Therefore, business process performance analysis has become a central issue in both academia and business, which aims at improving business processes by clarifying the characteristics and identifying the possible bottlenecks.

Traditional performance analysis tools (e.g., Business Objects and HP Business Process Cockpit) often focus on defined key performance indicators (KPIs), which are measured independent from process models [64]. In contrast, process mining techniques analyze business performance by projecting event logs onto process models, which intuitively present which parts of the process suffer problems. In this way, bottlenecks can quickly be identified and resolved.

Replay techniques [64, 137, 147] are commonly used to derive performance information based on a timed event log and an end-to-end process model. More precisely, these approaches assume that the start state is known, in order to replay each case of the log on the model. For one case, initially, a token is placed in the start place. Then, one by one, each transition (corresponding to the activity of the event in the case) in the model fires, thus consuming tokens from its input places and generating tokens in its output places. This is repeated until the case reaches a final marking. After replaying all cases, a collection of "token visits" are recorded for each place, and each token visit has a start and end time. Hence, a multi-set of durations can be derived, which are used to compute the waiting time. Similarly, each transition is also visited by tokens. If events in the log have start and complete lifecycles, we can also get the start and end time, which can be used to compute the service time. The replay approaches depend on procedural models, e.g., Petri nets, which have strict and clear semantics. Note that it is also possible to analyze performance based on other types of models, such as Fuzzy models [2, 56, 133] and directly-follows graphs (DFGs) [136]. For instance, Figure 8.1 presents a performance analysis result projected on a DFG in Disco, for a real-life application process of a personal loan in a Dutch Financial Institute (cf. BPI challenge 2012).

The already existing approaches explained above analyze performance based on process models, which describe the life-cycle of individual cases. This requires a case notion. However, a (global) case notion is missing in artifact-centric information systems, e.g., ERP/CRM systems, whose processes consist of interactive business functions such as procurement, production, sales, delivery,

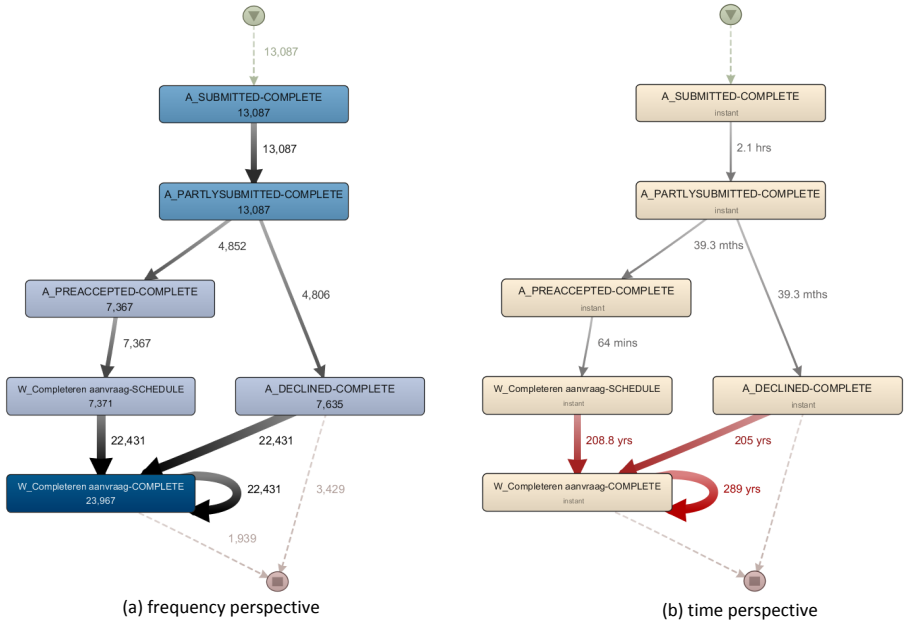


Figure 8.1: Performance analysis in Disco.

finance, etc. These processes are flexible and cannot be properly described by procedural models such as Petri nets (which assume a case notion for the whole process and have clear semantics for replaying). Besides, XES logs comprising process instances (which are taken as input for already existing approaches) are difficult to derive from artifact-centric information systems, as these systems are object-centric. In other words, they are built on top of database technology, i.e., the generated data contains hundreds of tables (covering customers, orders, deliveries, etc.) rather than explicit process instances.

To address the above problems (note that the problems are general problems and are not Disco-specific, cf. Section 8.4.3 for more details), in this chapter we analyze the performance of artifact-centric business processes based on OCBC models, which are more powerful to describe these processes (cf. Chapter 5). In general, techniques concerned with the analysis of processes cover several performance dimensions, e.g., time (how fast a process is executed), frequency (which part of a process is executed most frequently), cost (how much a process

execution costs), quality (how well the process meets customer requirements and expectations), etc. Here, we only focus on the time and frequency perspectives, like most performance analysis tools in process mining, e.g., the tool in Figure 8.1 (cf. Section 8.4 for more experiment results based on various tools). This chapter shows the advantages of OCBC models and XOC logs when it comes to performance analysis. We will show the following properties of OCBC-based performance analysis:

- We *remove the imprecision of performance results caused by complex relationships in data*. The data extracted from artifact-centric information systems consists of database tables, in which there exist one-to-many and many-to-many relationships. The XES log format requires a case notion to correlate events in these tables, resulting in logs with convergence and divergence problems (cf. Chapter 4). These problems lead to incorrectly reported frequencies (due to duplicated events) and times (events are wrongly correlated). In contrast, the XOC log format correlates events by the data perspective (i.e., objects), which can deal with the complex relationships in data and avoid the imprecision.
- *The performance analysis result is naturally displayed in one single diagram and bottlenecks are visualized intuitively*. Similar to the DFG in Figure 8.1, OCBC models can also present the performance of the whole loan process in one single diagram with highlighted bottlenecks. Note that the loan process involves multiple sub-processes, e.g., the activities whose names start with "A" and "W" are from different sub-processes. If there exist many-to-many relationships between sub-processes, the DFG in Figure 8.1 will suffer the imprecision problem explained above. A solution to avoid this problem is to analyze the performance for each sub-process, resulting in separate diagrams with disconnected performance information. In contrast, the OCBC model has no problems facing many-to-many relationships. In other words, the performance result can still be displayed in one single diagram in such a situation.
- *The performance analysis can become lightweight based on available domain knowledge*. Unlike replay techniques, which correlate and replay events (of a particular case) over the whole process model, OCBC models correlate events and analyze performance based on a particular correlation pattern. In other words, the performance analysis of the whole model can be split into independent smaller tasks for all correlation patterns. If one has the knowledge to identify some interesting parts of the process, one can compute the performance only for the patterns involved in these parts (rather than the whole model), such that the analysis takes less time.

8.2 Performance Analysis for the Time Perspective

In XOC logs, each event has a timestamp to indicate when the event happens. Based on these timestamps, we analyze the performance on the time perspective in this section. Note that although it is possible that objects can have attributes related to time (e.g., the time when an object is created, modified or deleted), we limit the performance analysis to activities (i.e., based on observed events).

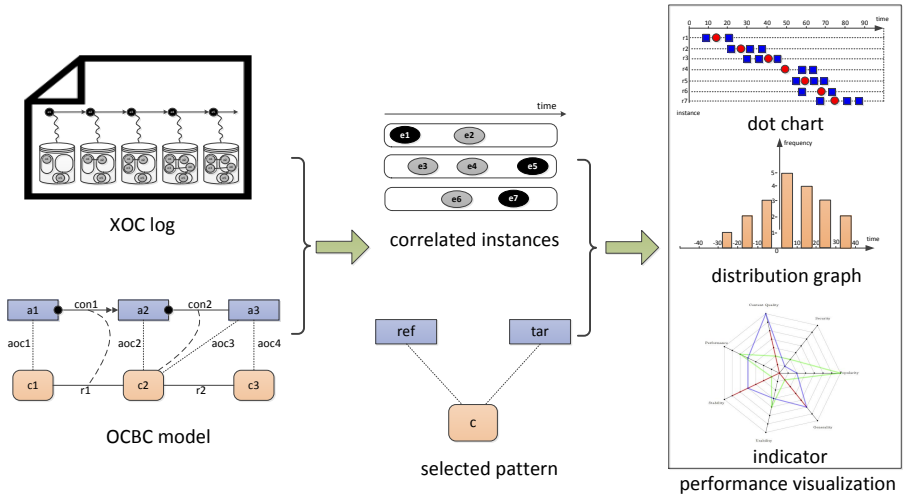


Figure 8.2: Analyzing and visualizing the performance on time perspective.

Figure 8.2 shows an overview on how to analyze performance based on an XOC log and an OCBC model. As events in XOC logs are not correlated explicitly and it makes no sense to analyze the time of individual events, we first correlate events based on correlation patterns extracted from the OCBC model (cf. Definition 6.8). More precisely, based on a selected correlation pattern, events are correlated, resulting in a set of pattern instances. Each instance contains one reference event and any number of target events. Then, the derived instances are analyzed and the performance is presented through three methods: dot charts, column charts and indicators. After deriving the performance result for each correlation pattern, it is possible to integrate them into a single diagram.

Correlating events using correlation patterns was explained in Chapter 6. Therefore, we illustrate how to analyze pattern instances with dotted charts, column charts and indicators.

8.2.1 Performance Analysis Using Dotted Charts

In any process mining project for performance analysis, the first intuitive idea is to get a general feeling of the data in event logs. Besides the various log inspectors in ProM which offer information such as the number of events and activities, the so-called dotted chart provides a helicopter view of the events in the log [130].

An event in an XOC log can have various attributes, such as “activity”, “time”, “resource”, “amount” and “customer”. In addition to the original attributes, correlated events have extra artificial attributes. As shown in Figure 8.2, events are correlated as instances based on a correlation pattern for performance analysis. During the correlation process, a new attribute “instance” is added to each event to indicate the instance it belongs to. Besides, an additional attribute “type” with two possible values “reference” and “target” is also added, because each instance comprises events of the reference or target activities of the correlation pattern. For example, the callout in Figure 8.3 describes the attributes of the second event in the last row. This event is of the reference activity and belongs to the seventh instance.

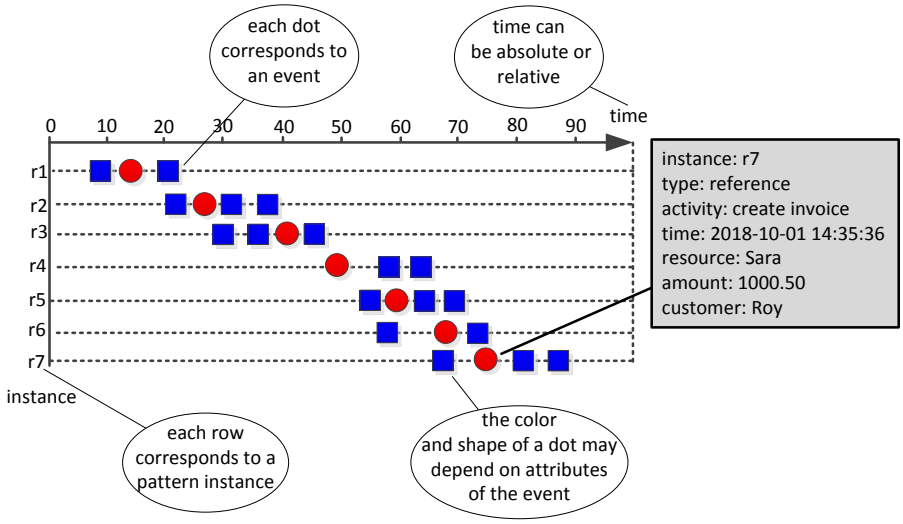


Figure 8.3: In a dotted chart, dots represent events, and the position, color, and shape of a dot indicate the attributes of the corresponding event.

In a dotted chart, each event is depicted as a dot in a two-dimensional panel as shown in Figure 8.3. In the panel, the horizontal axis represents the time of the event and the vertical axis represents the instance of the event. In other words, events are grouped in rows by instances and ranked by time. In each row, dots correspond to events from the same instance and the events on the left happen before the events on the right.

The color and shape of a dot can also indicate features of its corresponding event. Events which have the same value for a particular attribute can have the same color or shape. For instance, Figure 8.3 uses the color and shape to indicate attribute "type". Therefore, events with different "type" values have different colors and shapes, i.e., reference events are represented by red circles and target events are represented by blue squares.

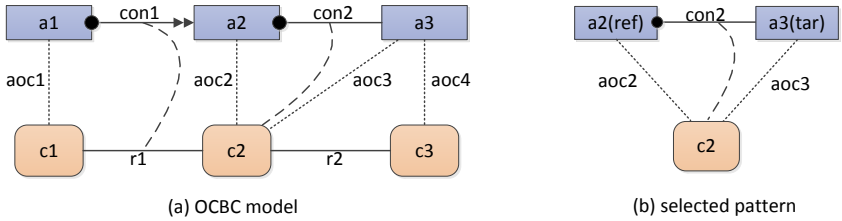


Figure 8.4: Selecting a correlation pattern from an OCBC model for performance analysis.

Next, we use an example to explain how to view performance using dotted charts. Assume that one is interested in the performance related to the correlation pattern in Figure 8.4(b), after analyzing the whole OCBC model in Figure 8.4(a). Based on the selected correlation pattern, events in the log are correlated, resulting in the instances in Figure 8.5. Note that at least one target event should happen before or after its corresponding reference event, indicted by the constraint $con2$. Figure 8.5 shows three instances on the left side, where the numbers in parenthesis indicate the event timestamps. For simplicity, we use integer timestamps to represent verbose timestamps like "2018-10-01 14:35:36". For instance, the reference event $e1$ happens at time 10. By selecting the instance as the class, each row in the dotted chart (on the right side in Figure 8.5) depicts a pattern instance.

Based on the dotted chart, it is easy to understand some interesting patterns. For instance, the target events $e2$, $e3$, $e6$ and $e7$ happen between time 15 and time 30, and $e5$ and $e9$ happen around time 60. If this rule is followed by thousands of instances, we can claim that it is a pattern (i.e., target events

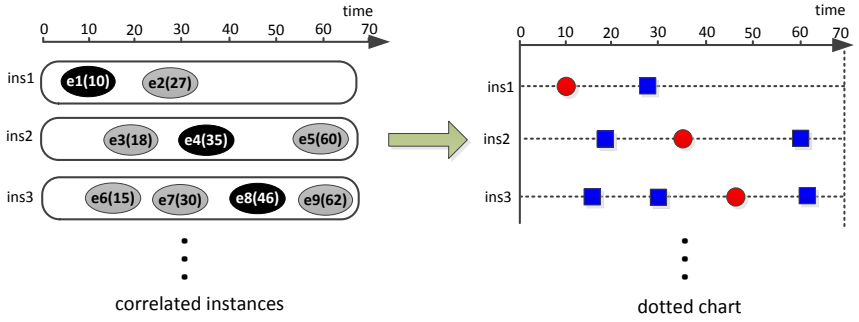


Figure 8.5: Presenting pattern instances with a dotted chart of absolute time.

happen in a batching way). In reality, this pattern can be interpreted as an insight that some types of events only happen in a fixed period of a day or a week, as the corresponding resources only work in that period.

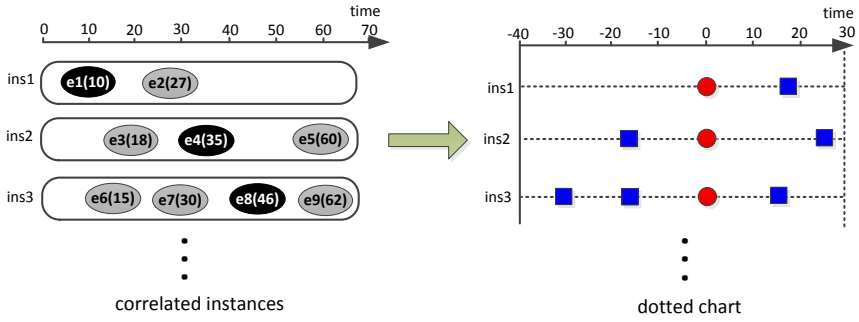


Figure 8.6: Presenting pattern instances with a dotted chart of relative time.

Note that the time dimension in a dotted chart can be absolute or relative. Figure 8.5 and Figure 8.6 map the same instances onto dotted charts with absolute time and relative time, respectively. In Figure 8.6, we place all reference events at time zero. Besides, each target event is placed based on its relative timestamp, i.e., the result of its absolute timestamp minus the absolute timestamp of its corresponding reference event. By comparing Figure 8.5 and Figure 8.6,

the dotted chart with relative time can be considered as the result of aligning all instances based on reference events in the horizontal dimension, which makes it easier to compare the target events before and after reference events.

The dotted chart is a very powerful tool to view events and analyze performance from different angles. One can see all events at once while potentially showing different perspectives at the same time (through class, time, color and shape). The dotted chart can be seen as an example of a visual analytics technique which leverages on the remarkable capabilities of humans to visually identify patterns, trends and irregularities in large datasets. Moreover, by zooming in one can investigate particular patterns. Note that it is possible that the horizontal axis represents an event attribute rather than time.

8.2.2 Performance Analysis with Column Charts

Dotted charts are used to derive a helicopter view of behavior. Based on the dotted chart, it is possible to count dots (i.e., events) in a time window (of a particular size), resulting in a column chart (or bar chart) which shows the frequency distribution of events in terms of time.

A significant parameter for deriving an interesting column chart is the size of the time window, which decides the density of columns. A too small or too large window size results in a column chart that provides little or no information. For instance, if the configured window size is too small, windows may contain just one or no event. In the derived column chart, the frequency of any column is one or zero, which does not tell anything about which columns are more frequent. In practice, this parameter can be customized by users. Figure 8.7 shows two column charts based on different window sizes. Given a window size, the time axis is divided into windows which are next to each other and do not overlap. For instance, given a window size of 10 time units, two window examples are $(0,10]$ and $(10,20]$. The first window includes all events which happen after time 0 and before or at time 10, and the second window includes all events which happen after time 10 and before or at time 20.

After identifying time windows, we can build a column for each window. The height of the column indicates the number of events in the corresponding window. Note that filtering techniques can be employed when counting events in a window. For instance, we can set a restriction on an attribute of events such that only events that have required attribute values are counted. In this way, the column chart is more flexible to present the frequency distribution of different kinds of events.

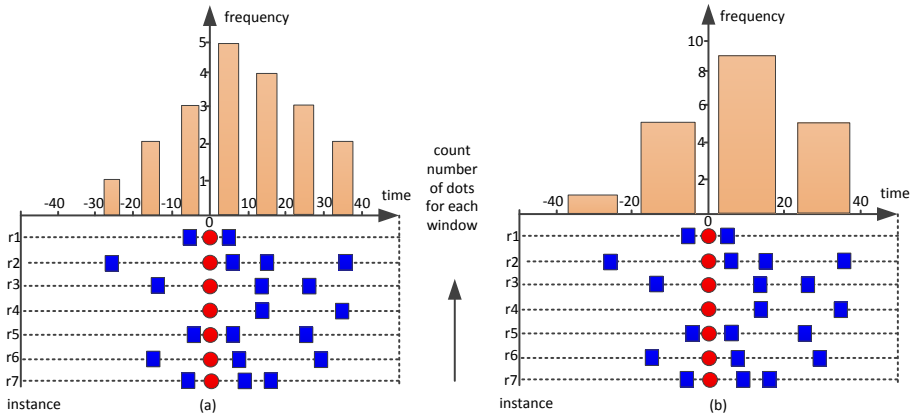


Figure 8.7: Two column charts based on window sizes of 10 (a) and 20 (b), respectively.

Figure 8.7(a) presents an example to show how to derive a column chart, based on a dotted chart with relative time and a window size of 10 time units. More precisely, in the dotted chart each row depicts a pattern instance, where blue dots represent target events and red dots represent reference events. The horizontal (time) axis in divided into ten windows based on the given window size, and vertical axis has numbers to indicate the frequency of events in each window. In this example, we only focus on the distribution of target events, as reference events are always located at time zero in terms of relative time. Therefore we only count the blue dots (target events) for each window. For instance, there are five blue dots in the window $(0,10]$ and four blue dots in the window $(10,20]$.

Column charts can provide intuitive insights to users. The column chart in Figure 8.7 shows the frequency distribution of target events in terms of relative time. Apparently, there are more target events after reference events than those before reference events. Besides, target events most likely happen within 10 time units after the reference event. The probability decreases when the timestamps of events are getting far from the window $(0,10]$. Based on the column chart, it is also possible to calculate the support of a customized window. For instance, the window $(-10,30]$ has a support of $(3+5+4+3)/(1+2+3+5+4+3+2) = 0.75$, which indicates that 75% of events fall into this window. Moreover, it is also possible to identify a window (of the smallest size) for a given support (there

can be many).

Note that if the number of columns is very large, we can zoom out on the time and connect the top of each column, resulting in a line chart. The line chart lifts the information on a higher level, which can visually indicate insights, such as patterns and trends.

8.2.3 Performance Analysis with Indicators

The performance of an organization can be measured through quantitative, measurable indicators. These are commonly known as Key Performance Indicators (KPIs). There exist various performance indicators, which can be time-related (e.g. throughput time of a process, service time of an activity), quality-related (e.g. visiting frequencies, error rates) or cost-related (e.g. process costs, material costs). For different purposes, different indicators are of importance.

As the idea of OCBC performance analysis is to divide the whole analysis into multiple smaller analysis tasks on the pattern level, in this section we propose some indicators related to correlation patterns. More precisely, similar to the throughput time of a process, we define durations of pattern instances to indicate its throughput time.

Definition 8.1 (Duration of Pattern Instance) *Let P be a correlation pattern and $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log. $\mathcal{U}_{TS}$ is the universe of timestamps. We assume that each event $e \in E$ has a timestamp, denoted as $\sharp_{time}(e) \in \mathcal{U}_{TS}$. Let $\mathcal{U}_{Dur}$ be the universe of time durations. Function $dur \in \mathbb{P}(E^*) \rightarrow \mathcal{U}_{Dur}$ returns the duration of a pattern instance.*

For a pattern instance $ins \in extI(L, P)$, we define the following versions for dur:

- *$durPre(ins) = \sharp_{time}(e_{ref}) - \sharp_{time}(e_f)$, computing the precedence duration of a pattern instance,*
- *$durRes(ins) = \sharp_{time}(e_l) - \sharp_{time}(e_{ref})$, computing the response duration of a pattern instance, and*
- *$durAll(ins) = durPre(ins) + durRes(ins) = \sharp_{time}(e_l) - \sharp_{time}(e_f)$, computing the total duration of a pattern instance,*

where $e_f = ins_1$ is the first event, $e_l = ins_{|ins|}$ is the last event and e_{ref} is the reference event in ins.[1]

Definition 8.1 defines the total duration, precedence duration and response duration for an instance corresponding to a correlation pattern. Intuitively, the

[1] ins_n means the $n\text{-}th$ event in the instance ins.

total duration of a pattern instance is the duration from its first event to its last event. Considering the reference event as a cut-off point, the total duration is divided into two segments, i.e., the precedence duration (before the reference event) and the response duration (after the reference event). Note that the precedence (response) duration is 0, if there are no target events before (after) the reference event.

Consider for example the instances in Figure 8.5. $durAll(ins3) = 47$, since its first event $e6$ happens at time 15 (i.e., $\sharp_{time}(e6) = 15$) and its last event $e9$ happens at time 62 (i.e., $\sharp_{time}(e9) = 62$). For the precedence duration, $durPre(ins3) = 46 - 15 = 31$ as the reference event $e8$ happens at time 46 (i.e., $\sharp_{time}(e8) = 46$). For the response duration, $durRes(ins3) = 62 - 46 = 16$, by subtracting the time of the reference event $e8$ from the time of the last event $e9$. Note that $durPre(ins1) = 0$ as there are no target events before the reference event $e1$.

In a pattern instance, there may exist multiple target events before and after reference events. Figure 8.8 shows a pattern instance *ins*, in which $e4$ is the reference event, and $e1$, $e2$ and $e3$ ($e5$, $e6$ and $e7$) are three target events before (after) the reference event. Definition 8.1 computes precedence (response) durations based on the first (last) target event, corresponding to the "longest" method in Figure 8.8. Based on this method, $durPre(ins) = 43 - 5 = 38$ and $durRes(ins) = 75 - 43 = 32$. It is also possible to compute the precedence (response) durations based on the closest target event before (after) the reference event, indicated by the "shortest" method. With this method, $durPre(ins) = 43 - 35 = 8$ and $durRes(ins) = 51 - 43 = 8$. Besides, we can use the median target event to compute the precedence (response) duration. For instance, as there are three target events before the reference event $e4$, $e2$ is the median target event. Similarly, $e6$ is the median target event for the events after $e4$. Based on the median events, $durPre(ins) = 43 - 14 = 29$ and $durRes(ins) = 58 - 43 = 15$. Moreover, we can calculate durations based on the mean time. The mean time for the events before (after) the reference event is $(35 + 5)/2 = 20$ $((75 + 51)/2 = 63)$. Therefore, $durPre(ins) = 43 - 20 = 23$ and $durRes(ins) = 63 - 43 = 20$.

Different methods can be applied to different situations in practice. Accordingly, the derived results are interpreted as different insights. Consider for example a pattern where "create order" is reference activity and "create shipment" is target activity. With the "longest" method, the calculated duration means that a customer receives all packages for his order after a time slot of the duration. In contrast, using the "closet" method, the duration means that the customer receives the first package after a time slot of the duration. In this thesis, we use the "longest" method by default.

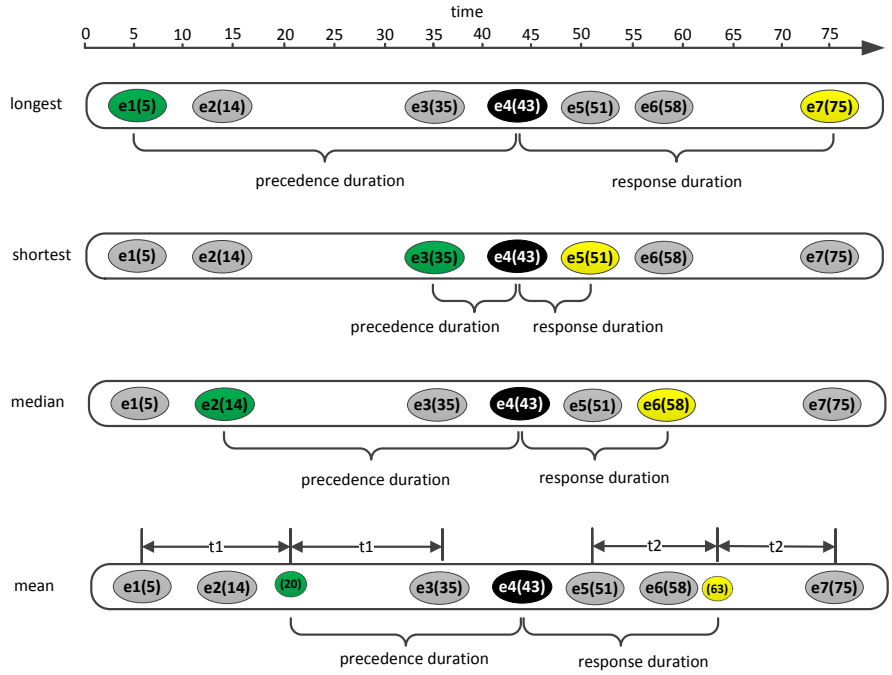


Figure 8.8: Various methods to compute precedence and response duration, where: black dots correspond to reference events and green (yellow) dots correspond to start (end) of instance durations.

Definition 8.2 (Average and Standard Deviation of Duration) *Let L be a sound XOC event log and P be a correlation pattern. Function $aveDur \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathcal{U}_{Dur}$ returns the average duration of pattern instances corresponding to P, such that*

$$aveDur(L,P) = \frac{\sum_{ins \in extI(L,P)} dur(ins)}{|extI(L,P)|}$$

where dur corresponds to $durAll$, $durPre$ or $durRes$ when computing the average of total duration, precedence duration or response duration, respectively. Function $devDur \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathcal{U}_{Dur}$ returns the standard deviation, such that

$$devDur(L,P) = \sqrt{\frac{\sum_{ins \in extI(L,P)} (dur(ins) - aveDur(L,P))^2}{|extI(L,P)|}}$$

Function *aveDur* computes the average duration of instances corresponding to a pattern by dividing the sum of durations by the number of instances. For example, the average of the total duration of instances in Figure 8.5 is $(17+42+47)/3 = 35.3$ as $durAll(ins1) = 17$, $durAll(ins2) = 42$ and $durAll(ins3) = 47$. Similarly, the average of the precedence duration is $(0+17+31)/3 = 16$ as $durPre(ins1) = 0$, $durPre(ins2) = 17$ and $durPre(ins3) = 31$, and the response average duration is $(17+25+16)/3 = 19.3$ as $durRes(ins1) = 17$, $durRes(ins2) = 25$ and $durRes(ins3) = 16$. Note that the precedence average duration plus the response average duration is equal to the total average duration.

Note that it is possible that two sets of instances have the same average duration but are quite different from each other. For instance, assume that the durations of the first set are 0, 30 and 60, and the durations of the second set are 30, 30 and 30. They have the same average duration, i.e., 30, but apparently the durations of the second set are more stable. In this case, we use the notion *standard deviation* of duration to reflect the difference, calculated by the function *devDur*. For instance, the deviation of the first set is $\sqrt{((0-30)^2+(30-30)^2+(60-30)^2)/3} = 24.5$ while the deviation of the second set is 0.

Based on the function *aveDur*, we can infer the average duration for each possible pattern of an OCBC model. Note that the average duration is the absolute duration and it only reflects the performance of a particular pattern individually. Often it is interesting to investigate durations on the whole model level (i.e., comparing durations of all patterns) and detect which part (i.e., pattern) consumes the most time. For this, we define the relative duration of a pattern by comparing its absolute duration with others.

Definition 8.3 (Relative Duration) *Let L be a sound XOC event log, $M = (ClaM, ActM, AOC, \sharp_A^{\square}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model and $P \in extP(ClaM, AOC)$ be a correlation pattern. Function $relDur \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ returns the relative average duration of pattern instances corresponding to P, such that*

$$relDur(L,M,P) = \frac{aveDur(L,P)}{\max_{P' \in extP(ClaM,AOC)} aveDur(L,P')}.$$

Definition 8.3 computes the relative duration for a correlation pattern, i.e., dividing its absolute average duration by the longest average duration (identified by comparing durations of all patterns). As shown in Definition 8.2, the average duration has three variants, i.e., the average of the total duration, the average of the precedence duration and the average of the response duration. Accordingly, there are three corresponding relative durations, i.e., relative total duration, relative precedence duration and relative response duration.

In order to explain how to compute relative durations, we present the following example. Assume that there exist four patterns $P1$, $P2$, $P3$ and $P4$ in total in an OCBC model, whose absolute precedence/response/total durations are 15/45/60, 70/30/100, 0/70/70 and 30/0/30, respectively, as shown in Table 8.1. For the precedence duration, 70 is the longest one. Divided by the longest duration, i.e., 70, the relative precedence durations are 0.21, 1.0, 0.0 and 0.43, as shown in the "Precedence duration (Relative)" column. In this way, we can also derive the relative response and total durations for these patterns. Note that the total duration is the sum of precedence and response durations. The "Total duration (Precedence ratio)" column indicates the ratio of precedence duration in the total duration. For instance, the precedence duration and total duration of $P1$ are 15 and 60, respectively. Therefore, the precedence ratio of $P1$ is $15/60 = 0.25$.

Pattern	Precedence duration		Response duration		Total duration		
	Absolute	Relative	Absolute	Relative	Absolute	Relative	Precedence ratio
P1	15	0.21	45	0.64	60	0.6	0.25
P2	70	1.0	30	0.43	100	1.0	0.7
P3	0	0.0	70	1.0	70	0.7	0.0
P4	30	0.43	0	0.0	30	0.3	1.0

Table 8.1: An example to explain how to compute relative durations.

After deriving durations, we propose a solution to project durations of all patterns onto an OCBC model, which supports humans to visually understand the performance on the time perspective. The idea of the solution is to highlight particularly long durations, i.e., the bottlenecks in the model. For the notation of durations in the model, we refer to the notation of behavioral constraints.

As shown in Figure 8.9(a), in an OCBC model a behavioral constraint (e.g., $con1$) is depicted as an edge with a black dot and an arrow between two activities, indicating the restriction between them. Each constraint has a corresponding correlation pattern, which is indicated by a dashed line (e.g., $cr1$). Similar to behavioral constraints, each duration also corresponds to a correlation pat-

tern. Therefore, we adapt the notation in Figure 8.9(a) to depict durations in Figure 8.9(b).

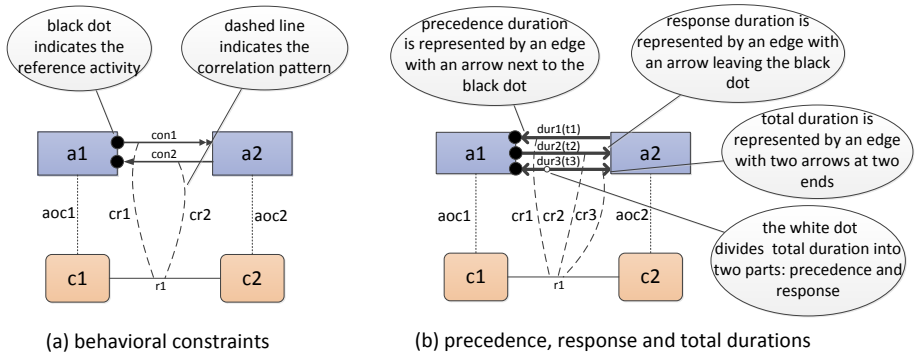


Figure 8.9: The difference of notations between behavioral constraints and durations.

We also use a black dot to indicate the reference activity, and a dashed line (e.g., $cr1$) to indicate the correlation pattern. Furthermore, the edges are employing barbed arrows for duration in Figure 8.9(b), which are different from compressed arrows for constraints in Figure 8.9(a). As shown in Figure 8.9(b), $dur1$, $dur2$ and $dur3$ refer to the precedence duration, response duration and total duration, respectively. In the parenthesis, we can indicate the absolute time (e.g., $t1$, $t2$ and $t3$) for the duration. For the precedence duration and response duration, the edges have only one arrow. When the arrow enters into (exits from) the black dot, the edge corresponds to precedence (response) duration. The total duration is represented by an edge with two arrows at both ends. Besides, there is a white dot in the edge to indicate the ratios of precedence duration and response duration. The part between the black dot and the white dot corresponds to the precedence duration while the other part corresponds to the response duration. For instance, if the ratio of precedence duration is 0.0, the white dot is next to the black dot. If the ratio is 0.5, the white dot is in the middle of the edge. Note that a duration is defined in the context of a correlation pattern and it is independent from behavioral constraints.

In order to give information about relative durations, we assign different colors to the arrows to highlight long durations, as shown in Figure 8.10, such that we assign dark red to long durations and light red to short durations. More precisely, by dividing the range of relative durations (i.e., $(0,1]$) into five intervals,

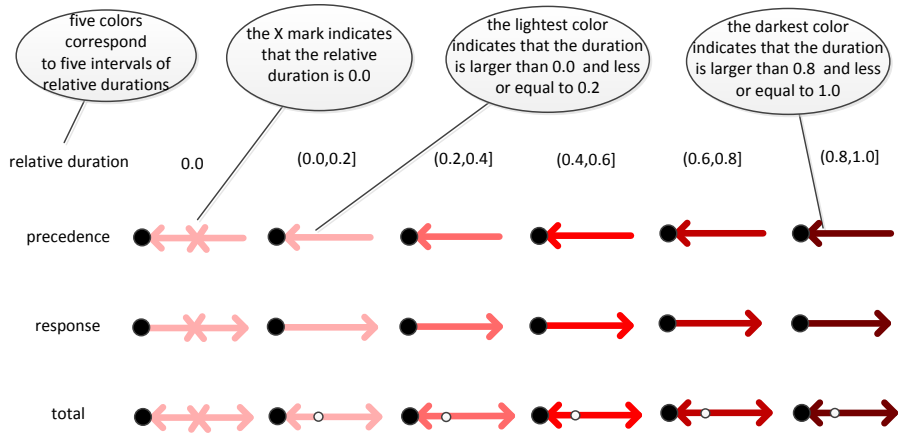


Figure 8.10: The color scale for relative durations.

five different degrees of red color are employed to depict them accordingly. For instance, the darkest red corresponds to $(0.8, 1.0]$ and the lightest red corresponds to $(0.0, 0.2]$. Note that it is possible that the relative duration is 0.0, which means that there are only reference events and no target events in pattern instances. In this situation, we add an X mark on the edge to indicate that the duration is 0.0. Based on the color solution explained above, any relative duration corresponds to precisely one edge in Figure 8.10. For instance, a precedence duration of 0.9 corresponds to the edge with the darkest red color (the rightest one) in the first row.

With highlighted long durations on the model, one can intuitively see the bottlenecks. We use an example to illustrate the approach of projecting durations introduced above. Figure 8.11 describes a process with an OCBC model, which is a part of the online shopping scenario. Before making an order of some products online, the customer often asks some questions about these products. After getting satisfying answers, the customer creates an order which contains some order lines (corresponding to the products). Then, the order lines are packed and shipped to the customer. In this process, there exist one-to-many and many-to-many relationships, e.g., (the order lines of) one order can be split into multiple shipments and one shipment can include order lines from multiple orders.

There are four possible correlation patterns $P1$, $P2$, $P3$ and $P4$ in total in

Figure 8.11. Their corresponding precedence, response and total duration are shown in Table 8.1. As the total durations contain information of both precedence and response durations, we only project them onto the model to make the model more readable without losing insights. As shown in the "Total duration (Relative)" column, the relative duration of $P2$ is 1.0. Therefore, its corresponding edge is in the darkest red in the model. Besides, the white dot is located on the edge based on its precedence ratio, i.e., 0.7. Similarly, we can project the total durations of other patterns onto the model. Note that if the precedence ratio is 0.0, the white dot is next to the black dot (e.g., $P3$) and if the precedence ratio is 1.0, the white dot is next to the target activity (e.g., $P4$).

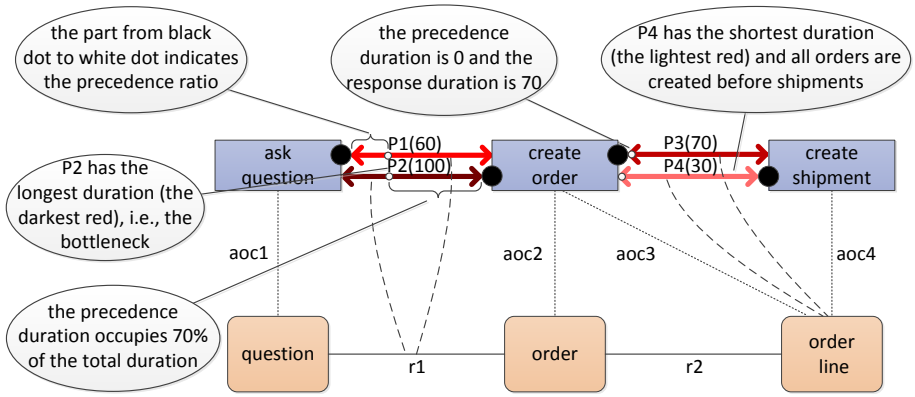


Figure 8.11: Highlighting bottlenecks in an OCBC model based on total durations in Table 8.1.

The projection of durations results in the OCBC model in Figure 8.11, which provides insights from the time perspective. Note that behavioral constraints will be hided in the OCBC model when presenting the performance, as the durations are related to correlation patterns rather than constraints.

For the correlation pattern $P2$, "create order" is the reference activity and "ask question" is the target activity. Accordingly, an instance corresponding to $P2$ reflects the consulting situation for a particular order. In real applications, customers can consult before orders (e.g., asking information about goods) and after orders (e.g., asking information about shipments or sales return). Indicated by the white dot, on average staff of the online shopping company spend more time (70%) answering questions before an order is placed than after it (30%).

Indicated by $P3$, all "create shipment" events happen after the corresponding "create order" events. After an order is created, on average the corresponding customer waits 70 time units until he receives the goods.

Up to now, we have introduced all approaches to analyze the performance on the time perspective. Different approaches are chosen based on different situations, and accordingly, the derived performance results are interpreted as different insights. Consider for example the pattern $P3$ in Figure 8.11. The "longest"/"shortest"/"median" method in Figure 8.8 to compute the response duration, indicates the amount of time it takes for the customer to receive his/her last/first/middle package for a specific order. Besides, the performance analysis in real applications can be split into three situations based on the available knowledge, explained as follows:

- If a user knows which pattern is most interesting, he/she can directly employ dotted charts and column charts to analyze the performance of the pattern.
- If the user has no domain knowledge at all, he can calculate the indicators of all patterns and project the performance onto the model. After highlighting patterns with long durations, the bottlenecks can be selected for further analysis with dotted charts and column charts.
- In most cases, the user has some knowledge to identify several interesting candidate patterns. For instance, in the process described in Figure 8.11, the patterns $P2$ and $P3$ are interesting candidates, as the user may want to know for an order, the time used to answer the related questions and the time used to deliver goods to the customer. In this situation, the indicators of all candidate patterns can be computed to detect bottlenecks, which are further analyzed with dotted charts and column charts. In this way, the performance analysis is faster, because we only spend time on analyzing interesting patterns rather than the whole model.

8.3 Performance Analysis for the Frequency Perspective

When analyzing performance of business processes, it is often interesting to compute how frequent each part of the model is actually used in real process executions. Based on the frequency information, it is possible to see the "highway" of the processes (which is the backbone of the processes) and the infrequent parts (which can be improved).

Most of the already existing performance analysis tools in process mining analyze the frequency perspective. However, they are focusing on the behavioral perspective, as the employed models do not support a powerful data perspective. In this section, we analyze the performance on the frequency perspective, and employ OCBC models to present the frequency information on the behavioral and data perspectives.

8.3.1 Mapping Frequencies onto OCBC Models

Based on an XOC log we derive the frequency information and project them onto an OCBC model. Furthermore, we classify the elements in OCBC models into five groups, i.e., classes (C), class relationships (R), activities (A), AOC relationships (AOC) and behavioral constraints (Con). Each element in the five groups has a frequency, indicating the number of its corresponding instances in the log.

Definition 8.4 (Frequency) *Let $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log and $M = (ClaM, ActM, AOC, \sharp_A^{\Box}, \sharp_A^{\Diamond}, \sharp_{OC}, crel)$ be an OCBC model where $ClaM = (C, R, \pi_1, \pi_2, \sharp_{src}^{\Box}, \sharp_{src}^{\Diamond}, \sharp_{tar}^{\Box}, \sharp_{tar}^{\Diamond})$ is the class model and $ActM = (A, Con, \pi_{ref}, \pi_{tar}, type)$ is the activity model. $Ele_M = C \cup R \cup A \cup AOC \cup Con$ is the set of all elements in the model. Function $fre \in Ele_M \rightarrow \mathbb{N}$ returns the frequency that an element $ele \in Ele_M$ is observed in the log. The function has five versions to calculate frequencies for the different types of elements.*

The frequency of a class is the number of objects of the class, i.e., for each $c \in C$,

$$freC(c) = |\{o \mid \exists e \in E : (o \in Obj_e \wedge class_e(o) = c)\}|.$$

The frequency of a class relationship is the number of object relations corresponding to the relationship, i.e., for each $r \in R$,

$$freR(r) = |\{(r, o_1, o_2) \mid \exists e \in E : (r, o_1, o_2) \in Rel_e\}|.$$

The frequency of an activity is the number of events of the activity, i.e., for each $a \in A$,

$$freA(a) = |\{e \in E \mid act(e) = a\}|.$$

The frequency of an AOC relationship is the number of references (between events and objects) corresponding to the relationship, i.e., for each $aoc = (a, c) \in AOC$,

$$freAOC(aoc) = |\{(e, o) \mid e \in E \wedge o \in relate(e) \wedge act(e) = a \wedge class_e(o) = c\}|.$$

The frequency of a behavioral constraint is the number of instances (derived based on P) which follow semantics of the constraint, i.e., for each $con \in Con$,

$$freCon(con) = |\{ins \in extI(L,P) \mid ins_{\downarrow P} \in type(con)\}|,$$

where $P = (\pi_{ref}(con), \pi_{tar}(con), crel(con))$ is the corresponding pattern of con.

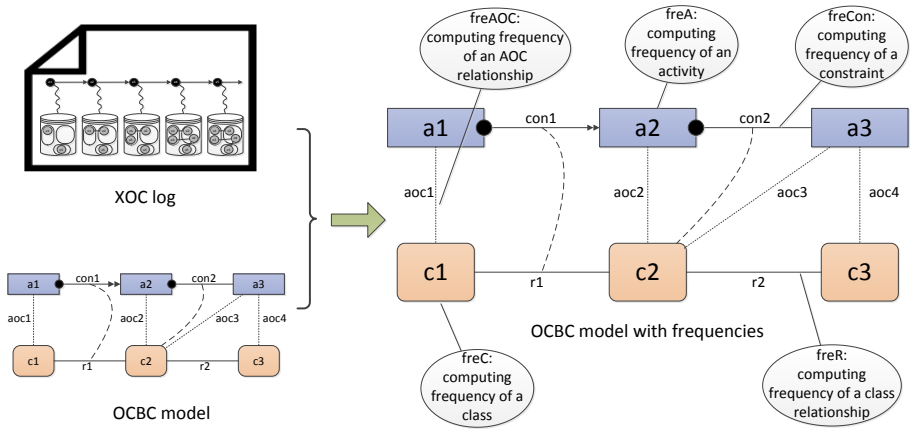


Figure 8.12: Computing the frequencies of different elements with *freC*, *freR*, *freAOC*, *freA* and *freCon*.

Definition 8.4 maps an XOC log onto an OCBC model in terms of the frequency perspective, as shown in Figure 8.12. It gives five functions to compute the frequencies of different types of elements. More precisely, function *freC* computes the frequency of a class, i.e., the number of all observed objects (in the log) corresponding to the class. Similarly, for a class relationship function *freR* computes its frequency by counting all corresponding object relations observed in the log. The frequency of an activity (indicated by *freA*) is the number of events corresponding to the activity. For an AOC relationship (a, c), its frequency (indicated by *freAOC*) is the number of reference relations between a events and c objects. Different from the previous frequencies, the frequency of a behavioral constraint is computed based on the events correlated by its corresponding pattern (i.e., pattern instances). For instance, assuming that P is the corresponding pattern of a behavioral constraint *con*, its frequency *freCon*(*con*) is the number of instances which satisfy the semantics of *con*.

Index	Event	Activity	References	Object Model	
				Objects	Relations
1	co1	co	o1,ol1,ol2	o1,ol1,ol2	(r10,o1,ol1), (r10,o1,ol2)
2	co2	co	o2,ol3,ol4	o1,ol1,ol2,o2,ol3, ol4	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4)
3	cs1	cs	ol1	o1,ol1,ol2,o2,ol3, ol4	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4)
4	ci1	ci	er1	o1,ol1,ol2,o2,ol3, ol4,er1	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4), (r8,o1,er1)
5	cs2	cs	ol1,ol2	o1,ol1,ol2,o2,ol3, ol4,er1	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4), (r8,o1,er1)
6	ci2	ci	er2,er3	o1,ol1,ol2,o2,ol3, ol4,er1,er2,er3	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4),(r8,o1,er1),(r8,o1,er2), (r8,o2,er3)
7	cs3	cs	ol3,ol4	o1,ol1,ol2,o2,ol3, ol4,er1,er2,er3	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4),(r8,o1,er1),(r8,o1,er2), (r8,o2,er3)
8	ci3	ci	er4	o1,ol1,ol2,o2,ol3, ol4,er1,er2,er3,er4	(r10,o1,ol1),(r10,o1,ol2),(r10,o2,ol3), (r10,o2,ol4),(r8,o1,er1),(r8,o1,er2), (r8,o2,er3), (r8,o2,er4)

Table 8.2: An XOC log example for computing frequencies.

Consider the log in Table 8.2 and the model in Figure 8.13 to understand how to compute the frequencies of different elements. *freC*(*element relation*) = 4 as there are four *element relation* objects in the log, i.e., $er1$, $er2$, $er3$ and $er4$ in the "Objects" column in Table 8.2. *freR*(*r8*) = 4 as there are four object relations of $r8$, e.g., $(r8, o1, er1)$ in the "Relations" column. *freA*(*create invoice*) = 3 as there are three *create invoice* events, i.e., $ci1$, $ci2$ and $ci3$ in the "Event" column. *freAOC*(*aoc4*) = 5 as there are five reference relations of $aoc4$ (between *create shipment* events and *order line* objects), i.e., $cs1$ refers to $ol1$, $cs2$ refers to $ol1$ and $ol2$, and $cs3$ refers to $ol3$ and $ol4$, indicated by the "Event" and "Reference" columns.

After explaining the frequency calculation of classes, class relationships, activities and AOC relationships, we show how to compute the frequencies of behavioral constraints. In order to do this, we need to correlate events first. Figure 8.14 shows the correlated events for $con1$, $con2$ and $con3$ based on the log in Table 8.2. We use $con2$ as an example to explain how to correlate events and compute the frequency. As shown in Figure 8.14(b), $cs1$ and $cs2$ are correlated to $co1$, because each of them has at least one object reference shared with $co1$. Similarly, $cs3$ is correlated to $co2$. Based on the correlated events and the order of events (indicated by the "Index" column in Table 8.2), we can derive two instances $\langle co1, cs1, cs2\rangle$ and $\langle co2, cs3\rangle$. The behavioral con-

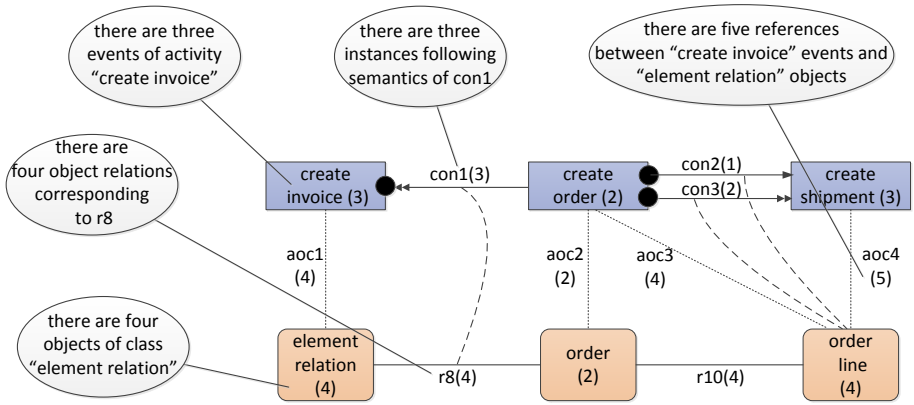


Figure 8.13: Computing the (absolute) frequencies based on the log in Table 8.2.

straint $con2$ requires that each reference event has precisely one following target event, i.e., $type(con2) = \{(before, after) \in \mathbb{N} \times \mathbb{N} \mid after = 1\}$. $ins_{\downarrow P}$ returns a pair of integers $(before, after)$, in which $before/after$ indicates the number of target events before/after the reference event in ins (cf. Definition 6.8). $\langle co1, cs1, cs2 \rangle_{\downarrow P} = (0,2)$ and $\langle co2, cs3 \rangle_{\downarrow P} = (0,1)$. Therefore, $freCon(con2) = 1$ as only the second instance satisfies $con2$.

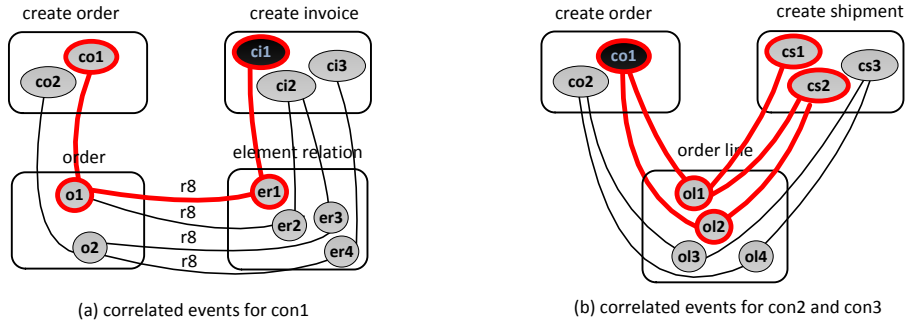


Figure 8.14: Correlating events to compute the frequencies of behavioral constraints.

Note that the frequencies in Definition 8.4 are absolute values. A typical

drawback of absolute frequencies is that the values always increase along with the size of the corresponding log. Therefore, it might be interesting to transform the absolute frequency into a relative frequency, i.e., a fraction between 0 and 1, which indicates the scale of the frequency compared with other frequencies.

Definition 8.5 (Relative Frequency) *Let Ele be the set of elements of a particular type, i.e., Ele is C, R, A, AOC, or Con. fre is the corresponding function to calculate the frequency. For each $ele \in Ele$, its relative frequency is*

$$relFre(ele) = \frac{fre(ele)}{\max_{ele' \in Ele} fre(ele')}$$

The relative frequency of an element ele in an OCBC model is the result of the absolute frequency of ele divided by the absolute frequency of the most frequent element of the same type as ele. Consider for example the absolute frequencies shown in Figure 8.13. For the AOC relationships, $aoc4$ has the largest frequency, i.e., 5. Divided by 5, the relative frequencies of $aoc1$, $aoc2$, $aoc3$ and $aoc4$ are 0.8, 0.4, 0.8 and 1.0, respectively.

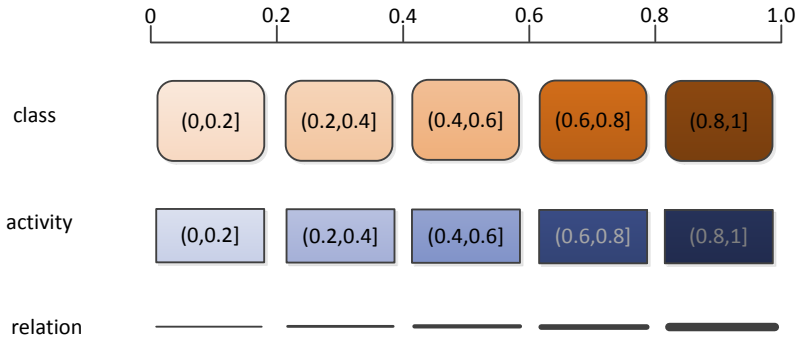


Figure 8.15: Color and width scale used for encoding class, activity and relation frequencies.

Based on relative frequencies, we can highlight the frequent elements of the model with outstanding colors for nodes or thick widths for edges, as shown in Figure 8.15. In general, the range of relative frequency (i.e., $(0,1]$) is divided into five intervals and each interval has a corresponding color or width. More

precisely, the *orange* and *blue* colors are used to paint classes and activities, respectively. A light color indicates a small relative frequency and a dark color indicates a large relative frequency. For instance, the lightest orange color indicates that the frequency of a class is in the interval of $(0, 0.2]$. Similarly, five different widths of edges are used to reflect the relative frequencies of relations and a thin (thick) edge indicates a small (large) relative frequency.

Figure 8.16 gives colors to classes and activities, and specifies the widths of relations based on the scale in Figure 8.15. For instance, the class "order" is colored in middle orange, which indicates that its relative frequency is in $(0.4, 0.6]$. In contrast, the activity "create invoice" is colored in the darkest blue, which indicates that its relative frequency is in $(0.8, 1.0]$. For the relations, $aoc4$ is the widest edge among all AOC relationships, which indicates that it is the most frequent. Clearly, the model with highlighted information provides the user with immediate feedback about the importance of each element in the model.

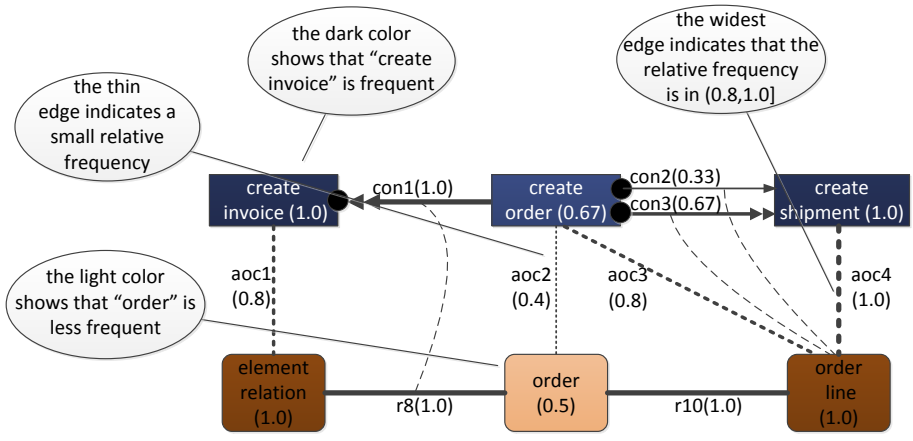


Figure 8.16: Highlighting frequent elements in the OCBC model to provide immediate feedback about their importance.

8.3.2 Computing Lengths of Pattern Instances

Section 8.3.1 proposed approaches to compute the absolute and relative frequencies of each element in an OCBC model, and projected the frequency information onto a model to highlight the frequent parts. In this section, we investigate

the frequency of target events in pattern instances, i.e., the lengths of pattern instances.

Definition 8.6 (Lengths of Pattern Instances) *Let P be a correlation pattern and $L = (E, act, attrE, relate, om, \preceq)$ be a sound XOC event log. Function $len \in \mathbb{P}(E^*) \rightarrow \mathbb{N}$ returns the length of a pattern instance. For a pattern instance $ins \in extI(L, P)$, we define the following versions for len:*

- *$lenAll(ins) = |ins| - 1$, computing the total length, i.e., the number of target events in the instance (subtracting 1 since each instance contains precisely one reference event),*
- *$lenPre(ins) = |ins_{\overleftarrow{P}}|$, computing the precedence length, i.e., the number of target events before the reference event,*
- *$lenRes(ins) = |ins_{\overrightarrow{P}}|$, computing the response length, i.e., the number of target events after the reference event,*

where $(|ins_{\overleftarrow{P}}|, |ins_{\overrightarrow{P}}|) = ins_{\downarrow P}$.

Definition 8.6 defines the total length, precedence length and response length of a pattern instance. Note that $ins_{\downarrow P} = (|ins_{\overleftarrow{P}}|, |ins_{\overrightarrow{P}}|)$ is a shorthand defined in Chapter 6, where $|ins_{\overleftarrow{P}}|$ ($|ins_{\overrightarrow{P}}|$) represents the number of the target events before (after) the reference event in a pattern instance. We use the three instances shown in Figure 8.17(a) to explain how to compute different lengths. For the total length, $lenAll(ins1) = 1$, as there is only one target event $e2$ in $ins1$. Similarly, $lenAll(ins2) = 2$ and $lenAll(ins3) = 3$. For the precedence length, $lenPre(ins1) = 0$ as there are no target events before the reference event $e1$. In the same method, $lenPre(ins2) = 2$ and $lenPre(ins3) = 1$. For the response length, there are one, zero and two target events after reference events in these three instances, respectively. Therefore, $lenRes(ins1) = 1$, $lenRes(ins2) = 0$ and $lenRes(ins3) = 2$

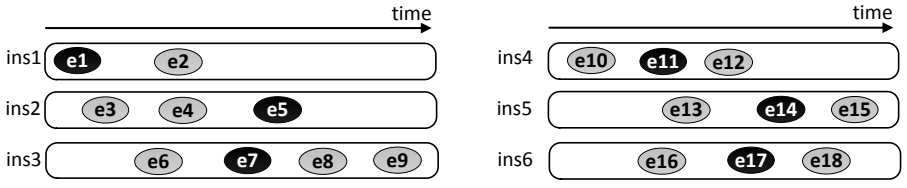


Figure 8.17: Two sets of pattern instances having the same average length but having different deviations.

Definition 8.7 (Average and Standard Deviation of Lengths) *Let L be a sound XOC event log and P be a correlation pattern. Function $aveLen \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathbb{R}$ returns the average length of pattern instances corresponding to P, such that*

$$aveLen(L,P) = \frac{\sum_{ins \in extI(L,P)} len(ins)}{|extI(L,P)|}$$

where len is function $lenAll$, $lenPre$ or $lenRes$ when computing average of total length, precedence length or response length, respectively. Function $devLen \in \mathcal{U}_L \times \mathcal{U}_P \rightarrow \mathbb{R}$ returns the standard deviation, such that

$$devLen(L,P) = \sqrt{\frac{\sum_{ins \in extI(L,P)} (len(ins) - aveLen(L,P))^2}{|extI(L,P)|}}$$

Based on the functions for computing the length of an individual instance, function *aveLen* derives the average length of all instances corresponding to a pattern, by dividing the sum of individual lengths by the number of instances. For example, the average of the total lengths of instances in Figure 8.17(a) is $(1+2+3)/3 = 2$, as $lenAll(ins1) = 1$, $lenAll(ins2) = 2$ and $lenAll(ins3) = 3$. Similarly, the average of the precedence (response) lengths is $(0+2+1)/3 = 1$ ($(1+0+2)/3 = 1$).

Note that the instances (*ins1*, *ins2* and *ins3*) in Figure 8.17(a) and the instances (*ins4*, *ins5* and *ins6*) in Figure 8.17(b) have the same averages of total length, precedence length and response length. However, it is obvious that they are quite different from each other. In this situation, the metric *standard deviation* can be used to reflect the difference, which is calculated by the function *varLen*. The deviation of the total lengths of the instances in Figure 8.17(a) is $\sqrt{((1-2)^2+(2-2)^2+(3-2)^2)/3} = 0.82$, while the deviation of the instances in Figure 8.17(b) is 0.

Note that the average length inferred by function *aveLen* is an absolute length. It only reflects the lengths of instances of a particular pattern. Often it is interesting to compare the lengths on the whole model level (i.e., in terms of all patterns) and detect which pattern has the longest instances.

Definition 8.8 (Relative Length) *Let L be a sound XOC event log, $M = (ClaM, ActM, AOC, \sharp_A^{\square}, \sharp_A^{\diamondsuit}, \sharp_{OC}, crel)$ be an OCBC model and $P \in extP(ClaM, AOC)$ be a correlation pattern. Function $relLen \in \mathcal{U}_L \times \mathcal{U}_{OCBCM} \times \mathcal{U}_P \rightarrow [0,1]$ returns the relative (average) length of pattern instances corresponding to P, such that*

$$relLen(L, M, P) = \frac{aveLen(L, P)}{\max_{P' \in extP(ClaM, AOC)} aveLen(L, P')}$$

Definition 8.8 computes the relative length for a correlation pattern, i.e., divides its absolute average length by the largest absolute average length (identified by comparing lengths of all patterns). As shown in Definition 8.7, the average length has three variants, i.e., average of the total length, average of the precedence length and average of the response length. Accordingly, there are three corresponding relative lengths, i.e., relative total length, relative precedence length and relative response length.

Pattern	Precedence length		Response length		Total length		
	Absolute	Relative	Absolute	Relative	Absolute	Relative	Precedence ratio
P1	1	0.25	2	0.67	3	0.6	0.33
P2	4	1.0	1	0.33	5	1.0	0.8
P3	0	0.0	3	1.0	3	0.6	0
P4	2	0.5	0	0.0	2	0.4	1.0

Table 8.3: An example to explain how to compute relative lengths.

In order to explain how to compute relative lengths, we employ the following example. Assume that there exist four patterns $P1$, $P2$, $P3$ and $P4$ in total in an OCBC model, and their corresponding precedence, response and total lengths are shown in Table 8.3. For the precedence length, $P2$ has the largest one (i.e., 4) and therefore its relative length is 1.0. The precedence lengths for $P1$, $P3$ and $P4$ are 1, 0, and 2, respectively. Divided by the largest length, i.e., 4, their relative precedence lengths are 0.25, 0.0 and 0.5, as shown in the "Precedence length (Relative)" column. Similarly, we can derive the relative response lengths and relative total lengths for these patterns. Note that the total length is the sum of precedence and response lengths. The "Total length (Precedence ratio)" column indicates the ratio of the precedence length in the total length. For instance, the precedence length and total length of $P1$ are 1 and 3, respectively. Therefore, the precedence ratio of $P1$ is $1/3 = 0.33$.

Similar to relative durations (cf. Section 8.2.3), we also propose a solution to project lengths of all patterns onto an OCBC model, highlighting the patterns which have longer lengths. The idea of the solution is to refer to the notation of durations (in Figure 8.10) but employing different degrees of the blue color (rather than the red color) to indicate the scale of lengths. As shown in Figure 8.18, we divide relative lengths into five intervals, in which the smallest

interval $(0.0, 0.2]$ corresponds to the lightest blue and the largest interval $(0.8, 1.0]$ corresponds to the darkest blue. Note that it is possible that the relative length is 0.0, which means that there are only reference events and no target events in pattern instances. In this situation, we again add an X mark on the edge to indicate that the length is 0.0. Based on the color solution explained above, any relative length corresponds to precisely one edge in Figure 8.18. For instance, a precedence length of 0.9 corresponds to the edge with the darkest blue color (the rightest one) in the first row.

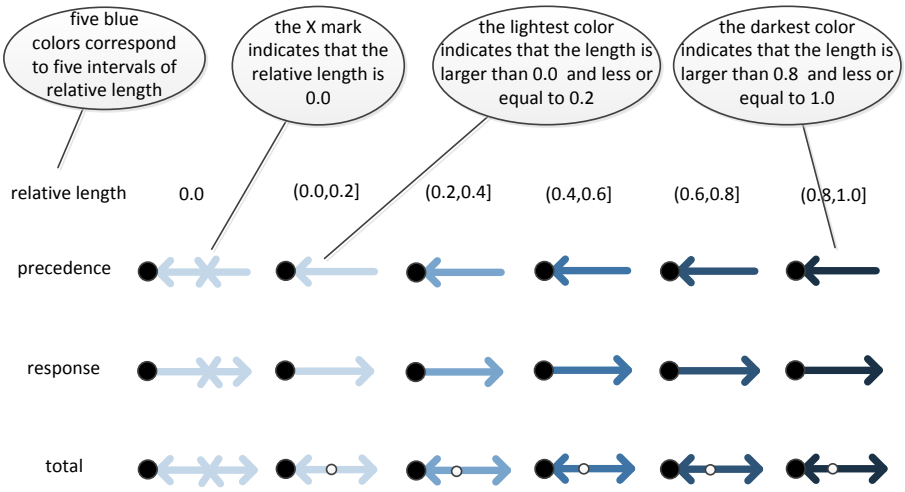


Figure 8.18: The color scale for relative lengths.

We use the same process in Figure 8.11 to explain how to project lengths onto models. There are four possible correlation patterns $P1$, $P2$, $P3$ and $P4$ in total. Their corresponding precedence, response and total lengths are shown in Table 8.3. Here, we only project the total lengths on the model to make the model easy to read. As shown in the "Total length (Relative)" column, the relative length of $P2$ is 1.0. Therefore, its corresponding edge is the darkest blue in the model. Besides, the white dot is located accordingly on the edge based on its precedence ratio, i.e., 0.8. Similarly, we can project the lengths of other patterns onto the model. Note that the white dot for $P3$ is next to the black dot as its precedence ratio is 0.0, and the white dot for $P4$ is next to the target activity as its precedence ratio is 1.0.

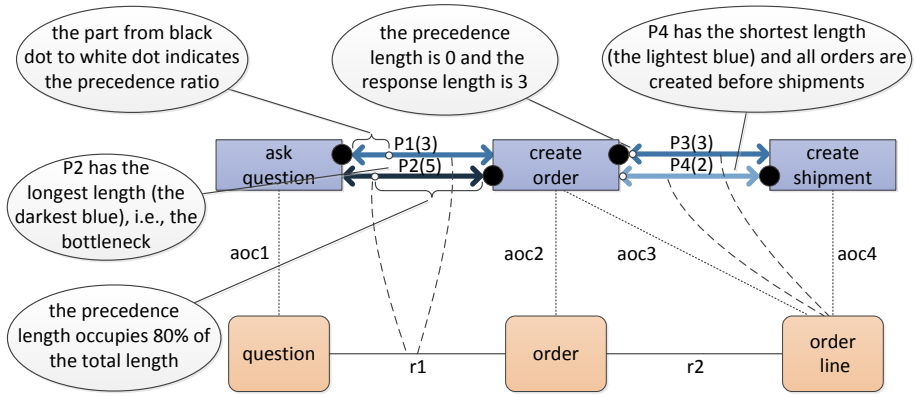


Figure 8.19: Highlighting bottlenecks in an OCBC model based on the total lengths in Table 8.3.

The projection of lengths results in the OCBC model in Figure 8.19, which provides insights in terms of instance lengths. For instance, the instances corresponding to $P2$ have the largest average length indicated by the darkest blue color, and the instances corresponding to $P4$ have the smallest average length indicated by the lightest blue color. For the correlation pattern $P2$, "create order" is the reference activity and "ask question" is the target activity. On average, each order has five related questions. As indicated by the position of the white dot, there are more questions before the order than those after the order. Besides, each order has on average three following shipments indicated by $P3$, and each shipment contains on average order lines from two orders indicated by $P4$.

8.4 Evaluation

In this section, we evaluate the OCBC performance analysis approach by comparing it with other performance analysis approaches while using the same raw event data. More precisely, based on the data generated in the OTC (order-to-cash) scenario of a real ERP system Dolibarr, different approaches are employed (our approach, but also more conventional approaches) to analyze the performance both on the frequency and time perspectives. Then, we compare the results derived by these approaches, to see which one can provide the best insights to

reveal the real performance.

8.4.1 Data Set

The OTC business process is a significant scenario in enterprises, which is supported by typical ERP systems. In this section, we introduce a data set generated in this scenario in an open source ERP system, Dolibarr.[2] The data set involves nine tables in the database of Dolibarr, as shown in Figure 8.20. There are ten relations (i.e., $r1$, $r2$,...,$r10$) between tables, which indicate the dependencies between records in different tables. For instance, the "order" column in the "order_line" table (a foreign key) refers to the "id" column in the "order" table (the primary key).

Note that the data set in Figure 8.20 is very small. Each table only contains one mandatory column (i.e., "id" column), and some other significant columns (e.g., "creation_date" and "customer" columns). Besides, all table records only cover transactions related to two orders ($o1$ and $o2$). The motivation for experimenting on such simple data is that we can use expertise to analyze the data and get insights about its performance. Then the analysis result derived manually can be considered as the ground truth to evaluate the results of different performance analysis approaches (i.e., the OCBC approach and other approaches explained later). Based on the ground truth, we can qualitatively compare approaches and identify problems.

We extract events from these tables. By considering the timestamp columns as activities, we identify four activities, i.e., "create order", "create shipment", "create invoice" and "create payment", which correspond to the "create_date" columns in tables "order", "shipment", "invoice" and "payment", respectively, as shown in Table 8.4. Events of different activities can be extracted from these four tables. For instance, an event $co1$ of activity "create order" with a timestamp "2017-08-11 10:33:37" can be derived based on the first record $o1$ in the "order" table. In this way, we extract ten events, as shown in the "Events" column of Table 8.4.

Besides extracting events, all records can be transformed into objects. Note that each record has a unique id shown in the "id" column, e.g., the first record in the "order" table has an id $o1$. By considering (i) the record id as the object id and (ii) the table of records as the class of objects, we can derive twenty-eight objects, as shown in the "Objects" column in Table 8.5. For instance, the object $c1$ corresponds to the first record in the "customer" table.

[2] https://www.win.tue.nl/ocbc/softwares/log_generation.html.

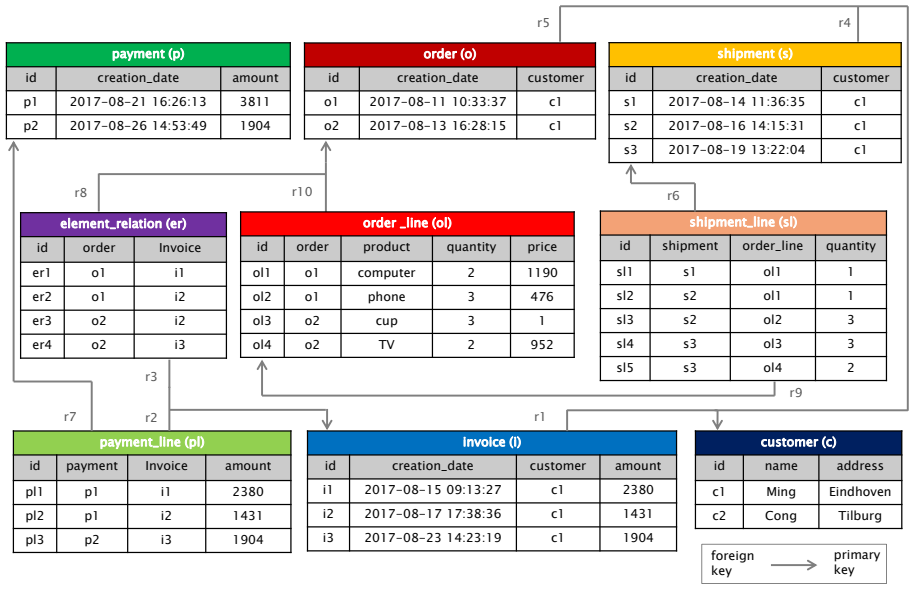


Figure 8.20: A small extraction of the database corresponding to the OTC business process in Dolibarr.

Table	Column	Activity	Abbreviation	Events
order	create_date	create order	co	co1,co2
shipment	create_date	create shipment	cs	cs1,cs2,cs3
invoice	create_date	create invoice	ci	ci1,ci2,ci3
payment	create_date	create payment	cp	cp1,cp2

Table 8.4: Identified activities and corresponding events.

Similarly, we can transform dependencies between records into object relations. For instance, as (i) object $o1$ corresponds to the first order record (in the "order" table), (ii) object $c1$ corresponds to the first customer record (in the "customer" table), and (iii) the order record refers to the customer record through the relationship $r5$, we get an object relation $(r5, c1, o1)$ between objects $c1$ and $o1$. Applying the same method, thirty-six object relations are derived as

shown in the "Relations" column in Table 8.5.

Index	Event	Activity	Timestamp	References	Object Model	
					Objects	Relations
1	co1	co	2017-08-11 10:33:37	o1,ol1,ol2	c1,c2,o1	(r5,c1,o1),(r10,o1,ol1),(r10,o1,ol2),(r5,c1,o2)
2	co2	co	2017-08-13 16:28:15	o2,ol3,ol4	ol1,ol2, o2	(r10,o2,ol3),(r10,o2,ol4),(r4,c1,s1),(r6,s1,sl1)
3	cs1	cs	2017-08-14 11:36:35	s1,ol1	ol3,ol4,s1	(r9,ol1,sl1),(r1,c1,i1),(r3,i1,er1),(r8,o1,er1)
4	ci1	ci	2017-08-15 09:13:27	i1,er1	sl1,i1,er1	(r4,c1,s2),(r6,s2,sl2),(r9,ol1,sl2),(r6,s2,sl3)
5	cs2	cs	2017-08-16 14:15:31	s2,ol1,ol2	s2,sl2,sl3	(r9,ol2,sl3),(r1,c1,i2),(r3,i2,er2),(r8,o1,er2)
6	ci2	ci	2017-08-17 17:38:36	i2,er2,er3	i2,er2,er3	(r3,i2,er3),(r8,o2,er3),(r4,c1,s3),(r6,s3,sl4)
7	cs3	cs	2017-08-19 13:22:04	s3,ol3,ol4	s3,sl4,sl5	(r9,ol3,sl4),(r6,s3,sl5),(r9,ol4,sl5),(r7,p1,pl1)
8	cp1	cp	2017-08-21 16:26:13	p1,pl1,pl2	p1,pl1,pl2	(r2,i1,pl1),(r7,p1,pl2), (r2,i2,pl2),(r1,c1,i3)
9	ci3	ci	2017-08-23 14:23:19	i3,er4	i3,er4,p2	(r3,i3,er4),(r8,o2,er4),(r7,p2,pl3), (r2,i3,pl3)
10	cp2	cp	2017-08-26 14:53:49	p2,pl3	pl3	

Table 8.5: The object-centric event data extracted from the database in Figure 8.20.

Table 8.5 presents the object-centric event data (cf. Chapter 3) extracted from the database in Figure 8.20. The extracted events form the behavioral perspective (indicated by "Index", "Event", "Activity" and "Timestamp" columns). The extracted objects and relations compose an object model, which represents the data perspective (indicated by "Objects" and "Relations" columns). The interactions between these two perspectives are indicated by the "References" column, which is explained next.

Here, we assume that each event refers to the record from which the event is derived. For instance, event $co1$ refers to the first record $o1$ in the "order" table (i.e., $co1$ refers to $o1$), as $co1$ is derived from $o1$. Based on domain knowledge, we know that each "create order" event also inserts several order lines (related to its created order) into the database. Therefore, $co1$ also refers to $ol1$ and $ol2$ as $ol1$ and $ol2$ are related to $o1$, as shown in the first row in the "References" column. In this way, we can derive all interactions between events and objects.

8.4.2 OCBC Performance Analysis Experiments

In this section, we employ the OCBC performance analysis approach to analyze the data set introduced in Section 8.4.1. We use the OCBC model in Figure 8.21 to present the performance information. The cardinalities on the class relationships and AOC relationships are hidden for simplicity, as they are not related to the performance analysis.

Based on the model, we extract six correlation patterns ($P1$, $P2$,..., $P6$) as shown in Table 8.6 (using function *extP* in Chapter 6). For example, $P1$ is an extracted pattern, in which "create order" (co) is the reference activity, "create

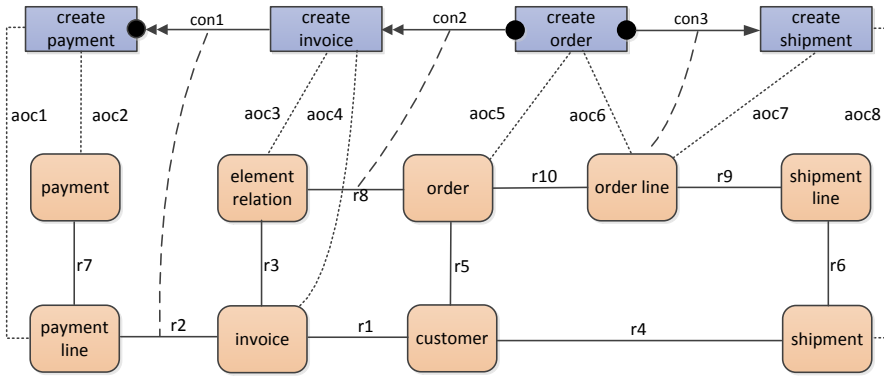


Figure 8.21: An OCBC model for presenting performacne information.

invoice" (ci) is the target activity and the class relationship $r8$ serves as the intermediary to connect these two activities.

Pattern	Reference activity	Target activity	Intermediary	Instances		
				Instance1	Instance2	Instance3
P1	co	ci	r8	$\langle co1, ci1, ci2\rangle$	$\langle co2, ci2, ci3\rangle$	-
P2	ci	co	r8	$\langle co1, ci1\rangle$	$\langle co1, co2, ci2\rangle$	$\langle co2, ci3\rangle$
P3	co	cs	ol	$\langle co1, cs1, cs2\rangle$	$\langle co2, cs3\rangle$	-
P4	cs	co	ol	$\langle co1, cs1\rangle$	$\langle co1, cs2\rangle$	$\langle co2, cs3\rangle$
P5	ci	cp	r2	$\langle ci1, cp1\rangle$	$\langle ci2, cp1\rangle$	$\langle ci3, cp2\rangle$
P6	cp	ci	r2	$\langle ci1, ci2, cp1\rangle$	$\langle ci3, cp2\rangle$	-

Table 8.6: Extracted correlation patterns and corresponding instances.

For each pattern, instances can be derived based on the data set in Table 8.5. Table 8.6 presents instances of all correlation patterns. Note that the real input for our performance analysis approach is an XOC log rather than an object-centric event data set. In an XOC log, each event has a corresponding object model to indicate the state of the database after the event. As the events in Table 8.5 do not remove objects, the object model corresponding to each event is a subset of the object model in the "Object Model" column. In other words, the "Object Model" column contains all objects observed in the process and it is not necessary

to rebuild the object model for each event. In this case, the object-centric event data can be considered as a simplified XOC log, which has enough information to generate instances for each pattern.

Pattern	Absolute Duration				Relative Duration				Precedence ratio
	Average	Deviation	Maximum	Minimum	Ave	Var	Max	Min	
P1	8d2h30m1s	2d13h24m5s	9d21h55m4s	6d7h4m59s	1.0	0.85	1.0	1.0	0
P2	6d17h13m17s	3d0h9m47s	9d21h55m4s	3d22h39m50s	0.83	1.0	1.0	0.62	1.0
P3	5d12h17m51s	0d12h9m40s	5d20h53m49s	5d3h41m54s	0.68	0.17	0.59	0.82	0
P4	4d16h32m53s	1d11h16m16s	5d20h53m49s	3d1h2m58s	0.58	0.49	0.59	0.48	1.0
P5	4d10h10m17s	1d16h34m1s	6d7h12m46s	3d0h30m30s	0.57	0.56	0.64	0.48	0
P6	4d15h51m38s	2d7h39m8s	6d7h12m46s	3d0h30m30s	0.55	0.77	0.64	0.48	1.0

Table 8.7: Durations for different patterns.

After recognizing instances for each pattern, we compute instance durations as shown in Table 8.7, based on the timestamps in the "Timestamp" column in Table 8.5. The average and deviation of instance durations are computed based on the method in Definition 8.2. Besides, we also identify the maximal and minimal durations for each pattern. Consider for example the pattern $P1$. The average duration is 8 days and 2 hours, and the deviation is 2 days and 13 hours. In all individual instance durations, the longest one takes 9 days and 21 hours, and the shortest one takes 6 days and 7 hours. By comparing the absolute durations of all patterns, the relative durations are derived based on the method in Definition 8.3. Note that the durations in Table 8.7 are total durations, and the "Precedence ratio" column indicates how much the precedence duration occupies in its corresponding total duration.

Figure 8.22 shows the performance information after projecting various durations in Table 8.7 onto the OCBC model in Figure 8.21. Note that for simplicity only the behavioral perspective of the OCBC model is presented. Based on the graphs, the performance of the whole process can be visualized intuitively. The numbers in parenthesis represent the absolute durations, the colors of edges indicate the relative durations, and the positions of white dots show the precedence ratios. Consider for example the pattern $P1$ in the "average" view to understand how to interpret the performance information:

- The darkest color indicates that $P1$ consumes the longest time during all patterns.
- The position of the white dot means that all target events (i.e., "create invoice" events) happen after the corresponding reference events (i.e., "create order" events).
- The number on the edge shows that on average a customer waits 8 days

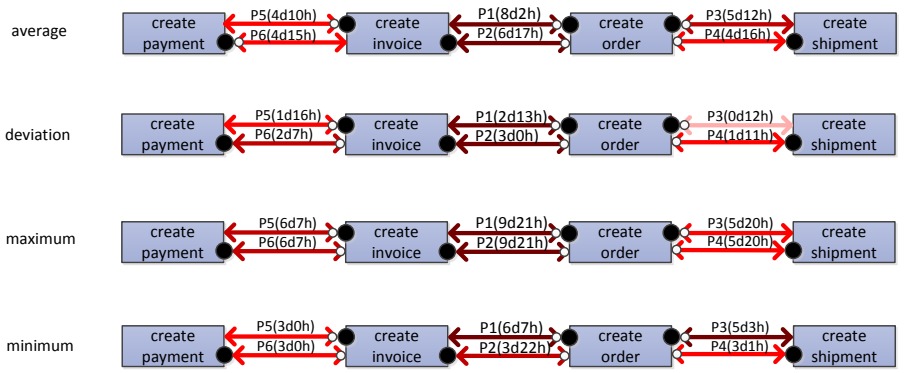


Figure 8.22: Projecting different durations onto OCBC models (only the behavioral perspectives of the OCBC model are shown for simplicity).

and 2 hours to receive all invoices from the company.[3]

In addition to the "average" view, other views can also provide significant insights. For instance, as shown in the "deviation" view $P1$ has the largest deviation, i.e., the durations between orders and invoices change a lot for different orders. In contrast, $P3$ has the smallest deviation, i.e., the waiting times for customers to receive shipments are stable. For the detected bottleneck, we can zoom in on its corresponding pattern instances with dotted charts and column charts for further analysis. As the data set is simple, here we do not present this part of the experiment.

After analyzing the time perspective, we shift to the frequency perspective. Figure 8.23 shows the frequencies of all elements in the OCBC model. As we can see in the model, "shipment line" is the most frequent class in all classes, which means that "shipment line" has the most objects observed in the process. On the behavioral perspective, activities "create invoice" and "create shipment" are more frequent than other activities. In real applications, a manager could assign more staff for these frequent activities.

[3]It is "all invoices" as durations are computed with the"longest" method by default. In contrast, it would be the "first invoice" if durations were computed with the"shortest" method.

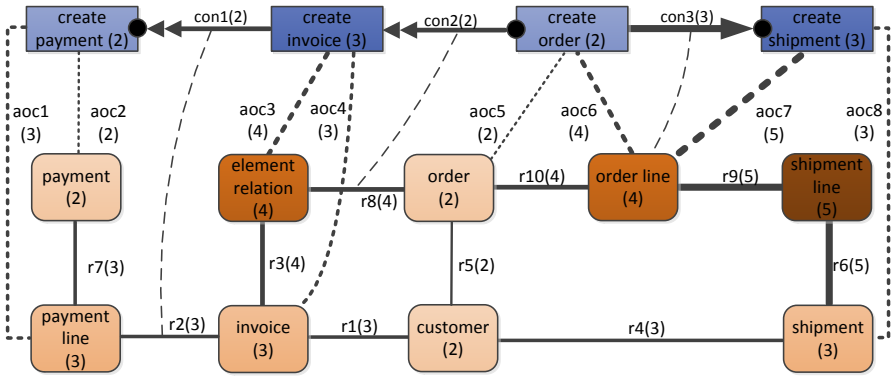


Figure 8.23: Projecting frequencies onto the OCBC model and highlighting frequent elements.

8.4.3 Comparison of Existing Process Analysis Tools

In Section 8.4.2, we applied the OCBC performance analysis approach to a data set from the OTC scenario in Dolibarr. In this section, other already existing process analysis approaches and tools are used on the same data set to derive performance results.

Based on the data set in Table 8.5, an XES log is extracted as shown in Figure 8.24. Note that the XES log has (i) convergence problems, e.g., $ci2$ is contained by two cases as if it is executed twice though it is performed only once in Dolibarr, and (ii) divergence problems, e.g., in case $case2$ "create payment" has two instances $cp1$ and $cp2$ which cannot be distinguished within the case, although they are performed on different documents in Dolibarr (i.e., $cp1$ is on $ci2$ and $cp2$ is on $ci3$).

Figure 8.25(a) presents the performance analysis result on the frequency perspective derived by Disco. More precisely, all frequency information is projected onto a DFG, which is discovered based on the directly-follow relations in the XES log. The numbers on the edges and nodes indicate the frequencies of relations and activities, respectively. For example, activity "create shipment" is observed three times, and "create invoice" is directly followed by "create payment" twice in the log. Note that some frequency information is not precise. For instance, "create invoice" happens 4 times which violate the reality, i.e., 3 times as shown in Table 8.5. Moreover, the DFG shows a misleading edge from "create payment" to

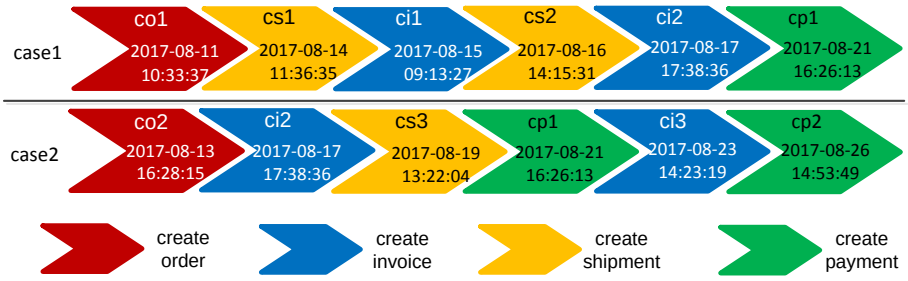


Figure 8.24: The XES log extracted from the data in Table 8.5.

"create invoice", since all "create payment" events happen after the corresponding "create invoice" events in real OTC processes. Note that these problems arise due to the convergence and divergence problems in the log. Disco is showing the right model given the flattened event log. In other words, the problems cannot be solved if XES logs are used as input for the tool.

Figure 8.25(b) presents the time information on the DFG. In the experiment, we choose "Mean duration" for the performance analysis, i.e., the numbers on the edges indicate the average times between activities. The interpretation of the numbers is simple and precise, in the situation with only one-to-one relationships between activities. For instance, "3.5 d" on the edge from "create invoice" to "create payment" means that after an invoice, on average we need to wait 3.5 days to receive the corresponding payment. However, if there are multiple payments for one invoice and multiple invoices for one payment (i.e., many-to-many relationship), the numbers are misleading and difficult to interpret (the 3.5 days are based on the two times that "create invoice" and "create payment" appear directly after one another in one of the cases). For instance, "3.5 d" can mean that after 3.5 days the first (or the last) payment for an invoice is received, when considering the invoice as reference. Besides, "3.5 d" can also be interpreted as that the time between the first invoice (or last invoice) and a payment is 3.5 days, when considering the payment as reference. Apparently, more than four interpretations can be made for the same number, which decreases the value of the insights. Traditional process discovery techniques showing frequencies and delays (like in Figure 8.25), face the following problems:

- *Not all useful information is employed to compute times*. After investigating how Disco computes the times for relations, we found that 3.5 days are the

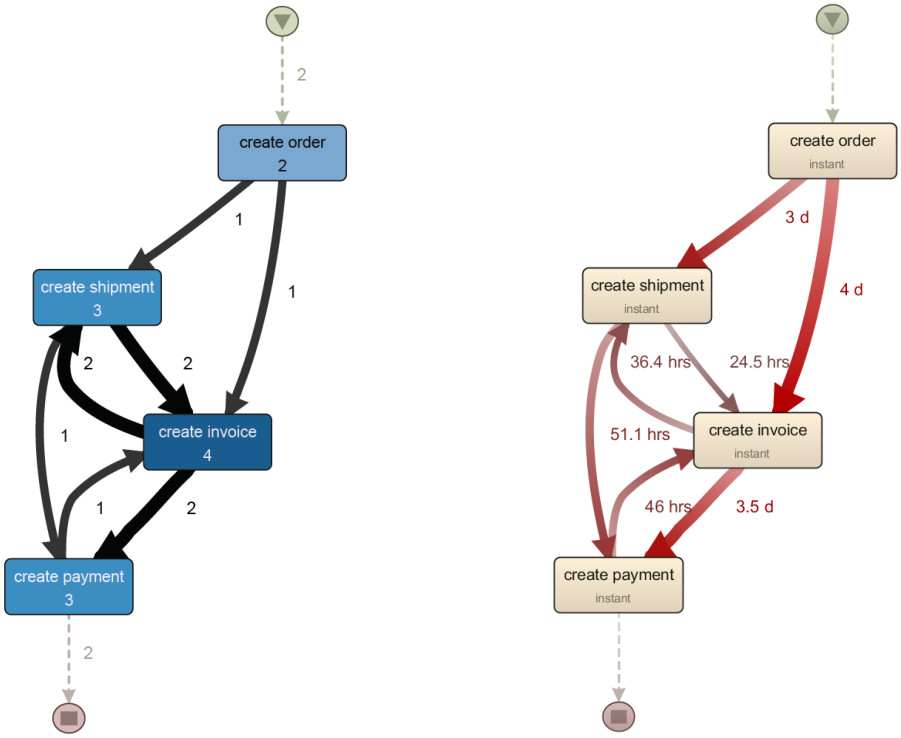


Figure 8.25: Analyzing the performance based on the log in Figure 8.24 with Disco.

average of the time between $ci2$ and $cp1$ (4 days) and the time between $ci3$ and $cp2$ (3 days). In other words, it only considers the times between directly linked events. Therefore, the time between $ci2$ and $cp1$ is discarded since they are not next to each other.

- *Interesting times can be missing.* If two activities are not observed next to each other in the log, e.g., "create order" and "create payment", there is no edge and time between them. In practice, users may be interesting in this time. A method to derive the time is to choose a path between them and sum all times on the path. However, the result with this method is not precise, as multiple paths can exist between two activities and the results

derived based on them can be quite different.

- *Times can provide misleading insights.* In the discovered DFG, the edges may conflict with real processes due to the convergence and divergence problems in logs. Accordingly, the times on these edges do not provide real insights. For instance, based on the number "46 hrs" on the edge from "create payment" to "create invoice", we can not claim that one customer receives a new invoice after he pays the last invoice.

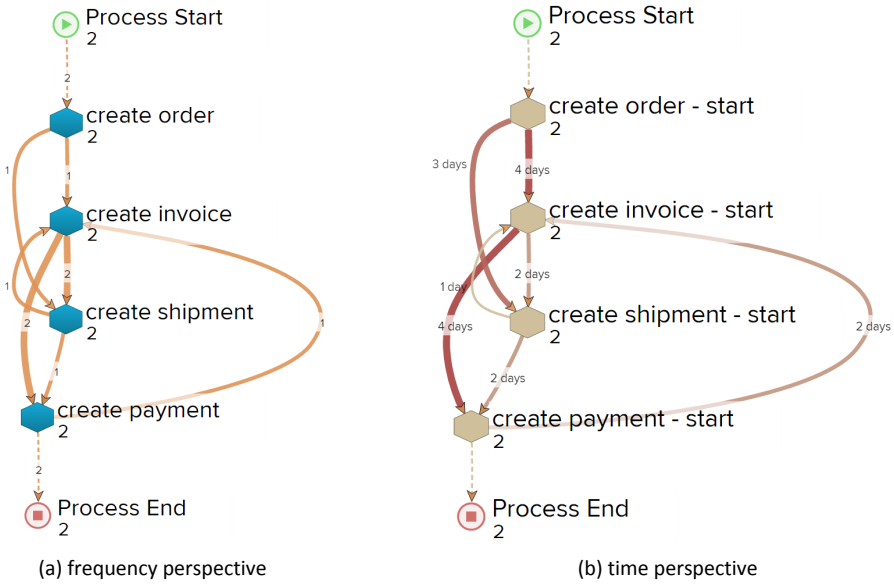


Figure 8.26: Analyzing the performance based on the log in Figure 8.24 with Celonis Process Mining.

We also used Celonis Process Mining (Version 4.0) to analyze the performance on the frequency perspective, resulting in the graph in Figure 8.26(a). All the activities have the same frequency, i.e., 2, which is inconsistent with the real frequency in the log in Figure 8.24. For instance, "create shipment" is observed 3 times, as shown in Table 8.5. The number on the edge from "create shipment" to "create invoice" is 1. However, this should be 2 since "create shipment" is directly followed by "create invoice" twice in the log. Figure 8.25(b) reveals the performance on the time perspective. The numbers on edges indicate the

average times between activities. In general, the derived graphs in Figure 8.26 are similar to the DFGs in Figure 8.25. They have the same nodes and edges, and slightly different numbers for frequencies and times. They suffer the same problems explained above for Disco.

Figure 8.27 displays a discovered Petri net with the Inductive visual Miner [79], setting parameters "activities" as 1 and "paths" as 1 (i.e., no filtering at all). The graph contains a lot of edges (and loops), and frequencies and times are presented on edges and nodes. The frequency number on an edge indicates the number of times the edge is visited by tokens. The frequencies of activities are indicated by the number of the entering (or exiting) edges. Besides, the numbers (on activities) correspond to sojourn times. Sojourn time is the sum of waiting time (the time between an activity becoming enabled and a resource starting to execute it) and service time (the time a resource is busy with a task). Since events in the log only have one lifecycle (rather than pairs of start and complete), the service time is zero. In other words, the sojourn time here represents the waiting time.

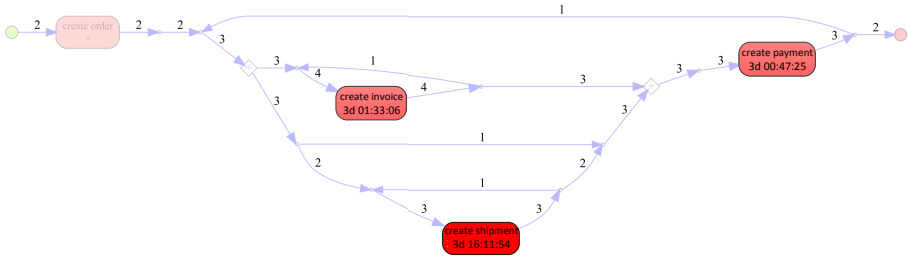


Figure 8.27: Analyzing the performance based on the log in Figure 8.24 with Inductive visual Miner (setting parameters "activities" as 1 and "paths" as 1).

Note that the times used to annotate the Petri net are more related to tokens rather than real waiting times between activities (i.e., different from the waiting times explained in a DFG). Consider for example the time t_{cs} on the "create shipment" activity, i.e., 3 days and 16 hours. It is the average of (i) the time $t1$ (3 days and 1 hour, between $co1$ and $cs1$), (ii) the time $t2$ (2 days and 2 hours, between $cs1$ and $cs2$) and (iii) the time $t3$ (5 days and 21 hours, between $co2$ and $cs3$). According to the definition of the waiting time on Petri nets, $t1$, $t2$ and $t3$ correspond to times between activity "create shipment" becoming enabled and a "create shipment" event happening. Note that based on the Petri net, activity

"create shipment" is enabled when a "create order" event, a "create payment" event or a "create shipment" event happens. $t1$ and $t3$ correspond to the times between a "create order" event and a "create shipment" event. In contrast, $t2$ corresponds the time between a "create shipment" event and another "create shipment" event. Therefore, the time t_{cs} cannot be interpreted as the waiting time between activities, such that after an order, a customer waits 3 days and 16 hours to receive the shipments. It reflects the waiting time related to resources, e.g., a shipment is ready to be created but it has to wait 3 days and 16 hours, because the resource is busy with other shipments.

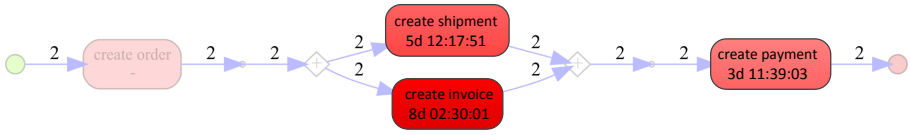


Figure 8.28: Analyzing the performance based on the log in Figure 8.24 with Inductive visual Miner (setting parameters "activities" as 1 and "paths" as 0.2).

As sojourn times are related to the structure of Petri nets, they will change when projecting the same log onto different Petri nets derived with different parameters. For instance, Figure 8.28 shows a discovered Petri net setting parameters "activities" as 1 and "paths" as 0.2. In this case, the discovered graph is simpler, as the filtering technique is employed. Note that the time on the "create shipment" activity is 5 days and 12 hours, which is quite different from the time (3 days and 16 hours) in Figure 8.27. The fact that performance result changes when setting different parameters makes the times on Petri nets more difficult to understand for users. Intuitively, the times, i.e., the performance result, should be consistent when users just want to get a brief view by filtering.

In comparison, the results derived by our OCBC performance analysis approach shown in Figure 8.22 and Figure 8.23 can more precisely reflect the real performance for the data set (with one-to-many and many-to-many relationships) in Figure 8.20. On the behavioral perspective, the performance information on the OCBC model presents frequencies of all elements and highlights the frequent parts. On the time perspective, the bottlenecks of the process are detected, and the derived times (i.e., durations) can be precisely interpreted as insights, which are easy to understand for users.

8.5 Related Work

Process mining techniques do not only discover process models and check conformance, but they also analyze performance of business processes. In this section, we review already existing performance analysis approaches in process mining [101].

Various approaches exist to analyze performance based on event logs. Most related to the first part of this chapter is the so-called dotted chart provides a helicopter view of an event log, which presents the performance on the frequency and time perspectives [130, 136]. More precisely, in a dotted chart each event is depicted as a dot in a two dimensional panel, where the horizontal axis represents the time of the event and the vertical axis represents the class of the event. By choosing different classes (e.g., attributes) as the vertical axis, we can analyze the performance from different perspectives. For instance, setting the vertical axis as cases, the dotted chart depicts the events of a case (along with time) in each row and provides some indicators related to cases, such as the average duration, the maximal duration and the minimal duration of cases. The dotted chart can be seen as an example of a visual analytics technique which leverages on the remarkable capabilities of humans to visually identify patterns, trends and irregularities in large datasets.

Most techniques provide performance information projected onto process models. In [147], an approach is proposed to obtain performance information by replaying a timed event log (i.e., an event log with timestamp information) on a workflow net (which is discovered from the log). This approach successfully obtains performance information of several time dimension metrics, such as flow times, waiting times, service times, synchronization times, etc. Consider two subsequent events $e1$ and $e2$, where $e1$ is of activity a and happens at "23-11-2011 15:56" and $e2$ is of activity b and happens at "23-11-2011 16:20". If b is causally dependent on a, we record a time of 24 minutes in-between a and b. By repeatedly measuring such time differences during replay, we can compute the average time that elapses in-between a and b. Unfortunately, this approach requires that the log fits the model perfectly, i.e., each case in the log should be a model instance. When the event log does not totally consist with the model, techniques for conformance checking, i.e., alignment, can be employed to align event log and model [137]. Moreover, [64] introduced an approach by enabling invisible transitions to fire such that the log does not have to fit the model.

Another highly related approach to performance analysis is based on Fuzzy models as proposed in [56]. [133] proposed a clustering algorithm to discover a Fuzzy model which only contains a limited set of nodes and edges. Based on the

model, two different ways were presented to project performance information onto it. As the model has relaxed semantics, the routing of activities needs to be estimated from encountered events in the event logs. For this purpose, a dedicated/relaxed log replay technique was developed for Fuzzy models [2]. Using these performance projections, one can gain insights of the process performance in an intuitive way.

However, none of the models (i.e., Petri nets and Fuzzy models) involved in the above approaches can handle processes that contain complex control flow dependencies such as cancellation, multiple concurrent instances, and advanced synchronization. They are thus insufficient to carry out performance analysis of processes which need such features. The YAWL language [145] was proposed to model the well-known workflow patterns, which supports complex control flow constructs such as multiple instances, cancellation regions and OR-joins. [116] demonstrated how performance analysis can be done for processes modeled in the YAWL language with such advanced concepts.

Often business process performance of a specific activity, case, or entire process highly depends on the context. For example, preceding activities, involved resources and their workload, or even the weather can have a big effect on performance. [85] presented a new methodological approach to identify the effect of contextual factors on performance in terms of processing time. This approach facilitates detecting impacted activities, i.e., activities within a business process that are indeed dependent on the context. [63] introduced a novel approach to analyze key process performance indicators by considering the process context. Each process entity was assigned a descriptive context label. Statistical hypothesis testing was used to verify whether a context label explains a significant difference in performance. Using this technique, the effect of any context on KPIs can be automatically analyzed and insights can be gained on root causes for delays, bottlenecks, deviations to protocol and violations of service level agreements, etc.

Most business process performance approaches offer various views showing the performance of a process over a given period of time, without considering how performance changes over time. [106] presented an approach to analyze the evolution of process performance via a notion of Staged Process Flow (SPF), which abstracts a business process as a series of queues corresponding to stages. If one knows how the temporal performance of a process evolves over a given period of time, he/she can take measures to avoid the possible bottlenecks. For instance, assuming that a bank manager knows how the waiting times in a loan application process evolved over the past month, he/she can adjust the resource allocation policies so as to minimize the effects of bottlenecks.

In real applications, most event logs only record either the start or the completion times of events, such that only the transition times between events are available and waiting or service times are not readily available. [108] proposed a novel method of estimating the average latent waiting and service times from the transition times, based on the optimization of the likelihood of the probabilistic model with expectation and maximization (EM) algorithms.

Next to the process mining techniques explained above, very few techniques are available to project performance information onto discovered process models. [64] compared the commercial process monitoring tools, which showed that (i) performance values are either measured based on a user-defined process model directly linking events in the log to parts of the model or (ii) they are measured totally independent from process models.

8.6 Summary

In this chapter, we analyzed the performance of artifact-centric business processes based on XOC logs and OCBC models. As OCBC models are more powerful to describe these processes, our approach can avoid the convergence and divergence problems when correlating events in the situation with one-to-many and many-to-many relationships, and derive precise performance analysis result.

Besides, the performance result is naturally displayed in one single diagram and bottlenecks are visualized intuitively. From the time perspective, we employ dotted charts and column charts to present the performance in more detail. Based on the defined indicators such as durations of pattern instances, the bottlenecks of the process are visualized intuitively after computing all durations and projecting them onto the model. From the frequency perspective, we compute the absolute and relative frequencies of each element in an OCBC model, and project this frequency information onto the model to highlight the "highways" of the process.

As OCBC models correlate events and analyze performance based on individual correlation patterns rather than the whole process, the performance analysis on the whole model can be split into independent smaller tasks for all correlation patterns. Based on this idea, users can implement customized performance analysis. If one has the knowledge to identify the interesting parts of the process, he/she can only compute the performance for the patterns involved in these parts to decrease the running time.

The limitation of our performance analysis is that it is time-consuming when the model is complex and there is no knowledge to identify some interesting correlation patterns, i.e., we have to compute the performance for all patterns.

Fortunately, as the whole performance analysis can be split into independent smaller tasks, the time-consuming problem can be solved by computing the smaller tasks with parallel techniques. This is a promising direction of future work.

Another future work could be correlating performance to the data perspective. In this chapter, performance is only computed for the behavioral perspective, e.g., the time it takes to deliver the goods after placing the respective order. As OCBC models and XOC logs support the data perspective as a first-class citizen, it is possible to link the performance result to objects, which can indicate the root cause of bottlenecks. For instance, a very long delivery time can be related to an order, a customer or a supplier, and the reason can be derived based on these objects, e.g., the order contains too many products, the customer lives too far away or the supplier is unreliable.

Chapter 9

Case Study of Stack Exchange

Up to now, we have introduced all OCBC techniques, such as model discovery, conformance checking and performance analysis. In this chapter, we present a case study based on a real-life data set from a Question & Answer website Stack Exchange, in which we show how these techniques are used in practice.

This chapter is organized as follows. In Section 9.1 we introduce the Stack Exchange website. Section 9.2 describes the data set and how to extract the raw data. Section 9.3 presents how to parse the data set, i.e., import the data set into database tables, and extract XOC logs from them. Based on the extracted logs, we discover OCBC models in Section 9.4. Taking an XOC log and an OCBC model as input, Section 9.5 checks conformance and Section 9.6 analyzes performance, respectively. Finally, Section 9.7 summarizes this chapter.

9.1 Introduction of Stack Exchange

Stack Exchange is a very popular Question & Answer internet community and a large repository of valuable knowledge. It has 174 child websites (counted on January 15, 2019), covering a wide variety of topics ranging from programming to cooking. Stack Overflow is a well-known child website of Stack Exchange, focusing on programming. Figure 9.1 shows some example topics, such as “Stack Overflow” and “Server Fault”. Each topic is represented by a message box, whose size indicates the number of questions related to the topic. Table 9.1 lists some children websites, each of which corresponds to a particular topic. More precisely,

the "#Question" column indicates the number of questions and the "Start time" column indicates the time when the first question was posted.

Name	Description	Start time	Website
Stack Overflow	for programmers	2008-07-31	stackoverflow.com
Mathematics	studying math at any level	2010-03-27	math.stackexchange.com
Ask Ubuntu	for Ubuntu users and developers	2009-01-08	askubuntu.com
Artificial Intelligence	artificial intelligence	2016-08-02	ai.stackexchange.com

Table 9.1: Details of some example child websites of Stack Exchange.

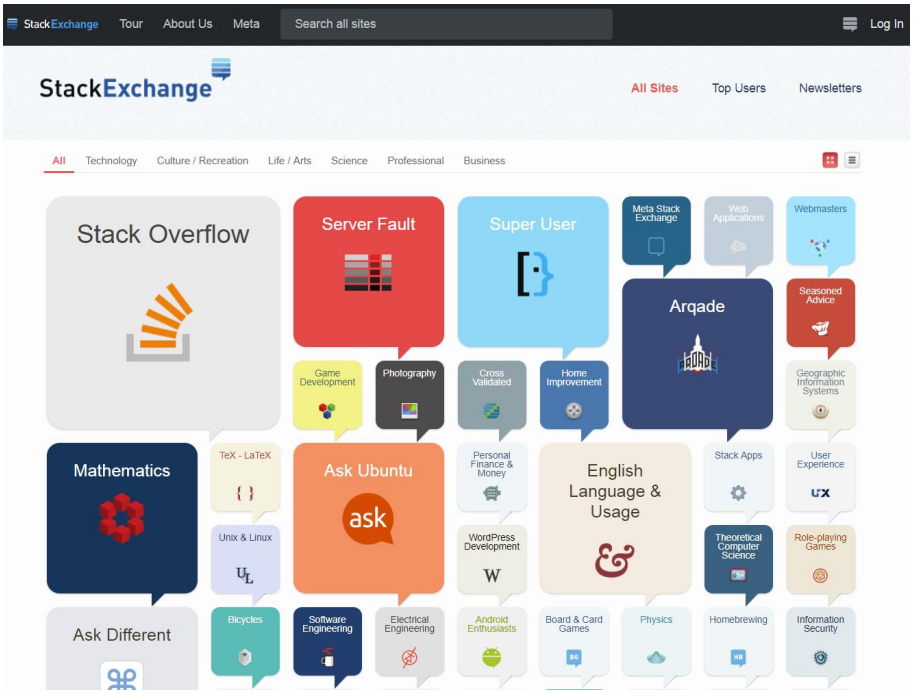


Figure 9.1: The topics in Stack Exchange.

Stack Exchange serves as a platform for users to ask and answer questions on emerging topics: users are expected to ask questions with simple factual answers and discussions are discouraged [29]. Besides, users can vote up or

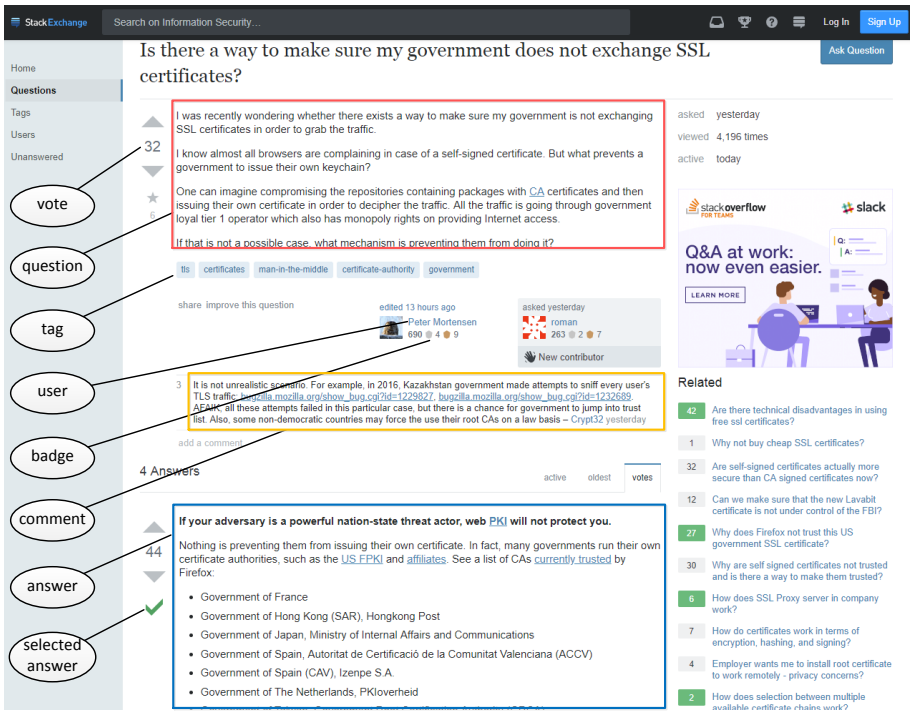


Figure 9.2: The interface of the Stack Exchange website.

down, comment and edit questions and answers in a way similar to a wiki or Digg. Figure 9.2 shows the interface of the website and the involved elements are explained as follows:

- ***Question***. The top part in the red square corresponds to the question.
- ***Comment***. The middle part in the orange square refers to a comment.
- ***Answer***. The bottom part in the blue square is an answer to the question. An answer can be identified as the "selected answer" by the user who posts the question, if he/she considers that the answer is the most relevant to the question. A question can have only one "selected answer", indicated by a check mark.
- ***Tag***. A tag is a word or phrase that describes the key content of the question. Tags sort questions into specific and well-defined categories, in order to

connect questions with experts who are able to answer these questions.

- ***Vote***. Users can vote up or down questions and answers. The number (e.g., 32) indicates the score of a question or an answer, derived by subtracting the count of "down" votes from the count of "up" votes.
- ***User***. People who want to post in the website must register as users. The user who posts a question or answer is shown under the question or answer.
- ***Badge***. Stack Exchange is a free service and users are not compensated for their services. However, users can earn reputation points and "badges" for their valued contributions. For example, a user is awarded 10 reputation points for receiving an "up" vote on an answer.

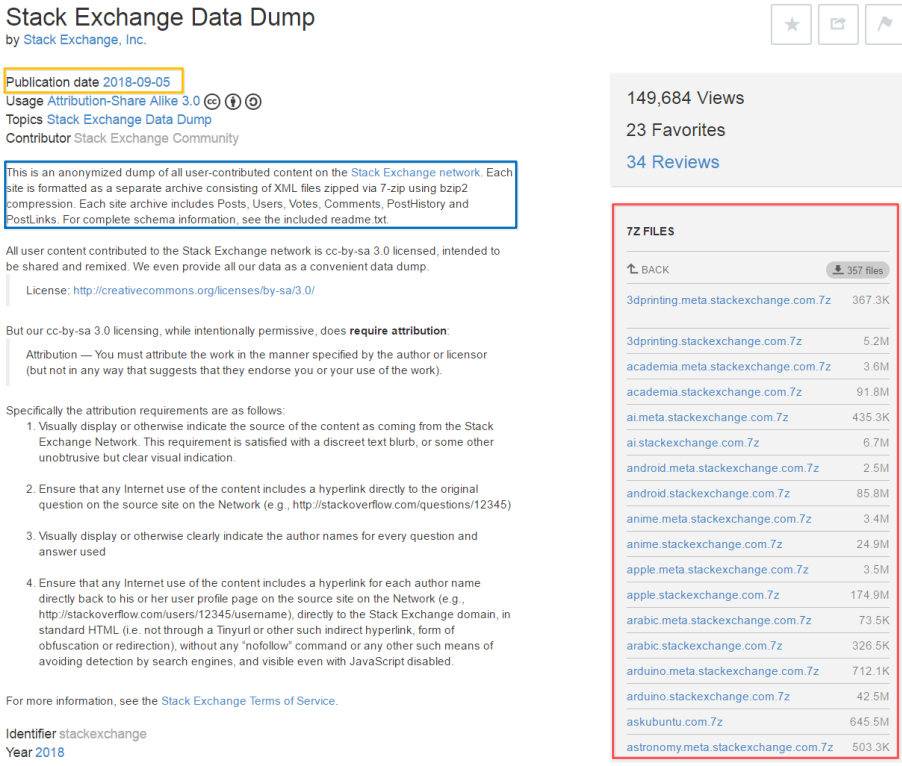


Figure 9.3: The Stack Exchange Data Dump.

9.2 Description of Data Set

Stack Exchange publishes its data regularly, in the form of a so-called Stack Exchange Data Dump. This enables access to historical data of Stack Exchange [29].

Figure 9.3 shows the interface where one can download the data dump. The time in the orange square indicates when the data dump is published. The paragraph in the blue square describes the data dump, i.e., it is an anonymized dump of all user-contributed content on the Stack Exchange network. The whole data dump consists of a bunch of children data sets, which are listed in the red square in Figure 9.3, and each of them corresponds to a topic, e.g., "ai.stackexchange.com.7z" corresponds to the "Artificial Intelligence" topic.

The data set used for our case study is a part of the data dump published on September 5th, 2018, corresponding to the "Artificial Intelligence" topic. It is formatted as a separate archive consisting of XML files zipped via 7-zip using bzip2 compression. After decompressing the 7-zip package, we derive eight files, "Badges.xml", "Comments.xml", "PostHistory.xml", "PostLinks.xml", "Posts.xml", "Tags.xml", "Users.xml" and "Votes.xml", as shown in Table 9.2.

Name	Description	#Instances	Size	Start time	End time
Badges.xml	badges assigned to users	14,515	1,592 KB	2016-08-02	2018-09-02
Comments.xml	comments for posts	7,102	2,308 KB	2016-08-02	2018-09-01
PostHistory.xml	histories of posts	17,246	15,095 KB	2016-08-02	2018-09-02
PostLinks.xml	links between posts	302	33 KB	2016-08-02	2018-08-31
Posts.xml	posted questions and answers	5,760	9,117 KB	2016-08-02	2018-09-01
Tags.xml	tags attached on the questions	287	24 KB	-	-
Users.xml	users registered in the website	15,541	6,770 KB	2016-08-02	2018-09-02
Votes.xml	votes for posts	22,338	1,979 KB	2016-08-02	2018-09-02

Table 9.2: Statistics of the data set corresponding to the "Artificial Intelligence" topic in Stack Exchange.

In other words, Table 9.2 provides a complete trace of all the actions on the "Artificial Intelligence" website of Stack Exchange for almost two years, i.e., from "2016-08-02" (when the website was founded) to "2018-09-02". The "#Instance" column indicates the count of instances for each file. For instance, there are 15,541 users in the "Users.xml" file. The "Start time" and "End time" columns show the earliest and latest timestamps found in each file, respectively, and their values are left empty when there is no timestamp column in the file, e.g., "Tags.xml".

Each file in Table 9.2 consists of a set of rows and each row corresponds to an

```
<?xml version="1.0" encoding="utf-8"?>
<badges>
  <row Id="1" UserId="4" Name="Informed" Date="2016-08-02T15:38:29.913" Class="3" TagBased="False" />
  <row Id="2" UserId="9" Name="Informed" Date="2016-08-02T15:39:20.227" Class="3" TagBased="False" />
  <row Id="3" UserId="16" Name="Informed" Date="2016-08-02T15:39:28.290" Class="3" TagBased="False" />
  <row Id="4" UserId="18" Name="Informed" Date="2016-08-02T15:39:40.377" Class="3" TagBased="False" />
  <row Id="5" UserId="10" Name="Informed" Date="2016-08-02T15:39:56.643" Class="3" TagBased="False" />
  <row Id="6" UserId="29" Name="Informed" Date="2016-08-02T15:44:49.750" Class="3" TagBased="False" />
  <row Id="7" UserId="30" Name="Informed" Date="2016-08-02T15:49:04.787" Class="3" TagBased="False" />
  <row Id="8" UserId="46" Name="Informed" Date="2016-08-02T15:50:04.000" Class="3" TagBased="False" />
  <row Id="9" UserId="5" Name="Informed" Date="2016-08-02T15:51:25.537" Class="3" TagBased="False" />
  <row Id="10" UserId="26" Name="Informed" Date="2016-08-02T15:52:06.490" Class="3" TagBased="False" />
  <row Id="11" UserId="4" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="12" UserId="8" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="13" UserId="9" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="14" UserId="15" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="15" UserId="16" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="16" UserId="18" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="17" UserId="19" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="18" UserId="20" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="19" UserId="26" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="20" UserId="29" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="21" UserId="33" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="22" UserId="38" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="23" UserId="44" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="24" UserId="48" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="25" UserId="50" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="26" UserId="51" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="27" UserId="52" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="28" UserId="53" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  <row Id="29" UserId="58" Name="Precognitive" Date="2016-08-02T15:56:06.070" Class="3" TagBased="False" />
  ...
</badges>
```

Figure 9.4: A fragment of the “Badges.xml” file.

instance which records all related attribute values. Figure 9.4 shows a fragment of the “Badges.xml” file. The first row corresponds to a badge instance which was created at “2016-08-02T15:38:29.913”. The id of the badge is “1” and it was given to a user whose id is “4”. The name of the badge is “Informed”, which indicates the level of the badge.

9.3 XOC Log Extraction

Section 9.2 described the data set derived from the Stack Exchange website. In this section, we extract XOC logs from the data set. As previously mentioned, the data set consists of eight XML files. Therefore, we must first parse and import them into database tables in order to be able to perform the log extraction.

In our case study, the key task is to explore the relations between questions and answers, thus the raw data must be parsed according to this goal. However, there are no explicit questions and answers in the raw data, since both questions and answers are considered as posts, stored in the same file “Posts.xml”. In order to get event logs containing objects and events related to questions and answers,

the posts need to be separated into two categories: questions and answers. We would like to have a separate class for questions and a class for answers.

Value	Name	Semantics	#Instances
1	Question	the post is a question	2,152
2	Answer	the post is a question	3,129
3	Wiki	the post is a community wiki	0
4	TagWikiExcerpt	the post holds the tag excerpt for a tag	238
5	TagWiki	the post holds the wiki text for a tag	238
6	ModeratorNomination	the post holds a nomination text provided by a moderator candidate	0
7	WikiPlaceholder	the post holds the text in the moderator election	3
8	PrivilegeWiki	the post is posted by a user who has a privilege	0

Table 9.3: The values and semantics of the attribute "PostTypeId" in the "Posts.xml" file.

Each post in the file "Posts.xml" has an attribute "PostTypeId", which has one of the eight different values shown in Table 9.3, indicating the type of the post. For instance, a post is considered as a question/answer if it has the value "1"/"2" for the attribute "PostTypeId". The "#Instances" column indicates the counts of posts of different types. For example, there are 2,152 questions and 3,129 answers, which cover the most posts. There are 479 posts of other types, which are excluded in the case study. In summary, we import questions and answers into "question" and "answer" tables, respectively, as shown in Table 9.4, and filter out other posts.

Similarly, we also need to separate comments into "question comments" and "answer comments", since all of them are stored in the same file "Comments.xml". As each comment refers to a question or answer through the attribute "postid", the "postid" value can be used for the separation. More precisely, if the "postid" value of a comment is the id of a particular question/answer, it is imported into the "question_comment"/"answer_comment" table. By applying the same method to the "Votes.xml" files, we derive another two tables "question_vote" and "answer_vote". Besides, the "User.xml", "Badge.xml", "PostHistory.xml", "Tag.xml" and "PostLinks.xml" files result in the "user", "badge", "history", "tag" and "post_link" tables, respectively.

Table 9.4 provides an overview of the eleven tables derived from the eight XML files in Table 9.2. The "Source file" column indicates the file where the table

Index	Name	Description	Source file	#Records	Class
1	question	posted questions	Posts.xml	2,152	question
2	answer	posted answers	Posts.xml	3,129	answer
3	question_comment	comments of questions	Comments.xml	3,803	q_comment
4	answer_comment	comments of answers	Comments.xml	3,299	a_comment
5	question_vote	votes of questions	Votes.xml	11,051	q_vote
6	answer_vote	votes of answers	Votes.xml	8,705	a_vote
7	user	registered users	Users.xml	15,541	user
8	badge	badges of users	Badges.xml	14,515	badge
9	history	operations on posts	PostHistory.xml	17,246	history
10	tag	tags of questions	Tags.xml	287	-
11	post_link	links between posts	PostLinks.xml	302	-

Table 9.4: Derived tables by parsing XML files in Table 9.2.

is extracted while the "#Records" column shows the number of records in each table. In order to extract XOC logs, we assign a class to each table, for example, the "question_comment" table corresponds to the "q_comment" class. Note that since the "tag" and "post_link" tables have few records, we do not assign classes to them, and so they are excluded in the case study.

Based on the common knowledge in terms of the relations between tables [103], Figure 9.5 presents the data schema of the derived tables shown in Table 9.4. "question" table has a foreign key "FK1" (corresponding to the "owneruserid" column) which references the primary key "PK" (corresponding to the "id" column) of the "user" table. Note that the table "history" has multiple foreign keys, where "FK1" references to "PK" in the "question" table, "FK2" references to "PK" in the "user" table and "FK3" references to "PK" in the "answer" table.

The records in the tables explained above can be transformed into objects, in other words, into the data perspective of the resulting XOC logs. From this perspective, a "badge" record in the "badge" table with an id of "1" can be transformed into an object of class "badge" with an id of "badge1". In addition to objects, it is possible to extract events from the database tables, forming the behavioral perspective of the resulting XOC logs. The idea is to consider the timestamp columns in tables as activities. The "badge" table has a timestamp column "date" (corresponding to the timestamp attribute "Date" in the file "Badge.xml" in Figure 9.4), which can be considered as an activity "get_badge". To summarize the process, Table 9.5 shows twelve possible activities derived from the database tables.

After mapping timestamp columns into activities, events can be extracted from database tables. Based on the key relations, we can derive the reference rela-

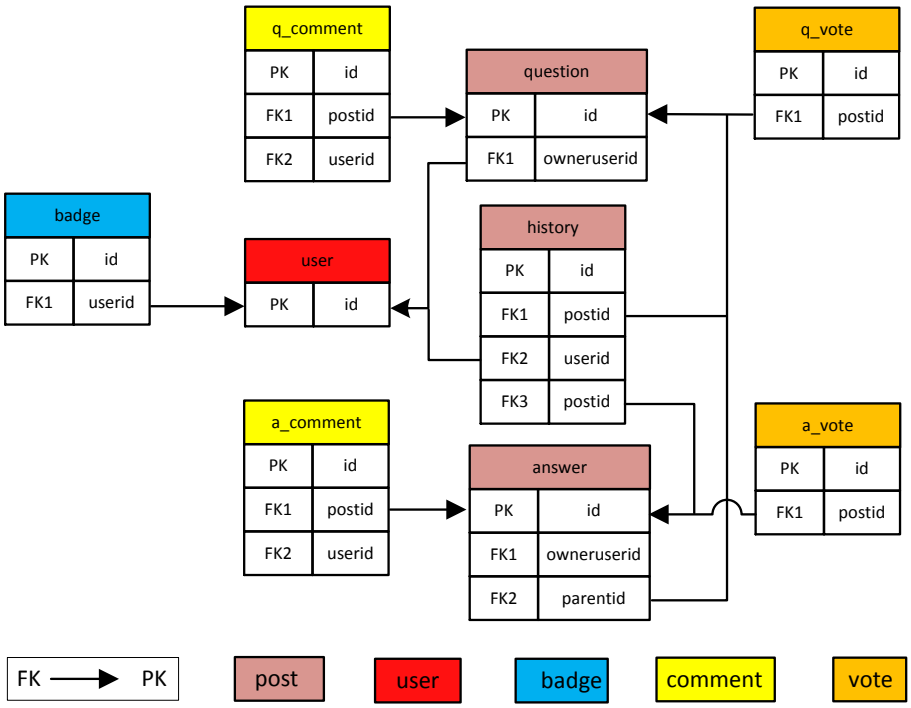


Figure 9.5: The data schema of the derived database tables shown in Table 9.4.

tions between events and objects/records. The value “2016-08-02T15:38:29.913” in the first record of “badge” table (corresponding to the first row in the file “Badge.xml” in Figure 9.4) can be transformed into a “get_badge” event which happened at 15:38:29 on August 2nd, 2016 and refers to the object “badge1”.

Up to now, we finished parsing the data set from the Stack Exchange website and extracting XOC logs from it. The extracted logs are used for analysis in the next sections. Models can be discovered as a result from this analysis, and to understand this output in an easy way we could filter out some activities when extracting a specific XOC log. The column “Included in log” in Table 9.5 is used for this purpose, i.e., when it is set to “No”, the event will not be included in the extracted log. In the example in Table 9.5, four different events will be filtered out: “access_last”, “close_question”, “edit_question_last”, and “edit_answer_last”.

Index	Table	Timestamp column	Activity	Included in log
1	user	creation_date	register	Yes
2	user	last_access_date	access_last	No
3	question	creation_date	post_question	Yes
4	question	close_date	close_question	No
5	question	last_edit_date	edit_question_last	No
6	answer	creation_date	post_answer	Yes
7	answer	last_edit_answer	edit_answer_last	No
8	question_comment	creation_date	make_question_comment	Yes
9	answer_comment	creation_date	make_answer_comment	Yes
10	question_vote	creation_date	vote_for_question	Yes
11	answer_vote	creation_date	vote_for_answer	Yes
12	badge	date	get_badge	Yes

Table 9.5: Mapping between activities and table columns

Note that it is also possible to filter based on the extracted XOC logs. However, the beforehand filtering can decrease the time used for extracting XOC logs.

9.4 OCBC Model Discovery

Taking the XOC logs generated in Section 9.3 as input, we discover OCBC models in this section, and in addition, we investigate the distribution of cardinality constraints. Note that with different parameters, the discovery technique can return different models. Here, we require the discovery technique to return a model where (i) its fitness is 1, (ii) it only contains positive behavioral constraints and (iii) the discovered cardinalities are normalized (cf. Chapter 6). Note that the above setting just discovers a model to provide some basic insights. It is possible to discover a model using other parameters or repair the discovered model based on known knowledge (cf. Section 9.5).

Figure 9.6 shows the discovered OCBC model based on the requirements mentioned above. In a single diagram the data and behavioral or control-flow perspectives are described (bottom and top, respectively), as well as the interplay between them (middle). It clearly reveals the involved classes (e.g., "question"), activities (e.g., "post_question") and constraints in the Question & Answer process in the Stack Exchange website, which is explained next.

The main task of the class model discovery (cf. Chapter 6) is to derive the cardinality constraints on the relation between classes. By taking $r6$ as an example, the cardinality constraint $\Box 1$ is discovered at its "question" side. It means that each answer always corresponds to precisely one question. Actually, the cardinality constraint $\Diamond 1$ is also discovered, which means that each answer

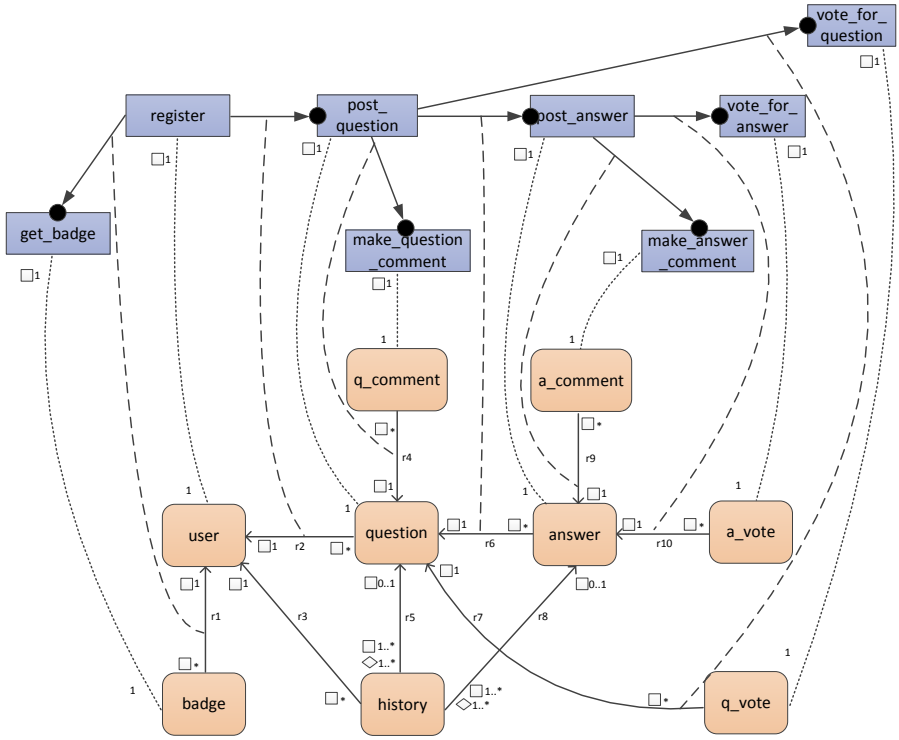


Figure 9.6: The discovered OCBC model which describes the Question & Answer process in the Stack Exchange website.

eventually corresponds to precisely one question. It is omitted in Figure 9.6 since it is implied by $\square 1$. Similarly, $\square *$ and $\Diamond *$ are discovered at the "answer" side on $r6$ where $\Diamond *$ is also omitted. They indicate that each question can correspond to any number of questions.

The panel shown in Figure 9.7 presents the distribution for the discovered cardinality constraints on class relationships. Diagrams simply show the distribution of the cardinalities of $r6$, taking question as a reference. There are four drop-down menus at the top of the panel, which are used to configure the distribution. In Figure 9.7, reference value is set as "question", target as "answer" and relation as "question-parentid-answer" (corresponding to $r6$). It means that

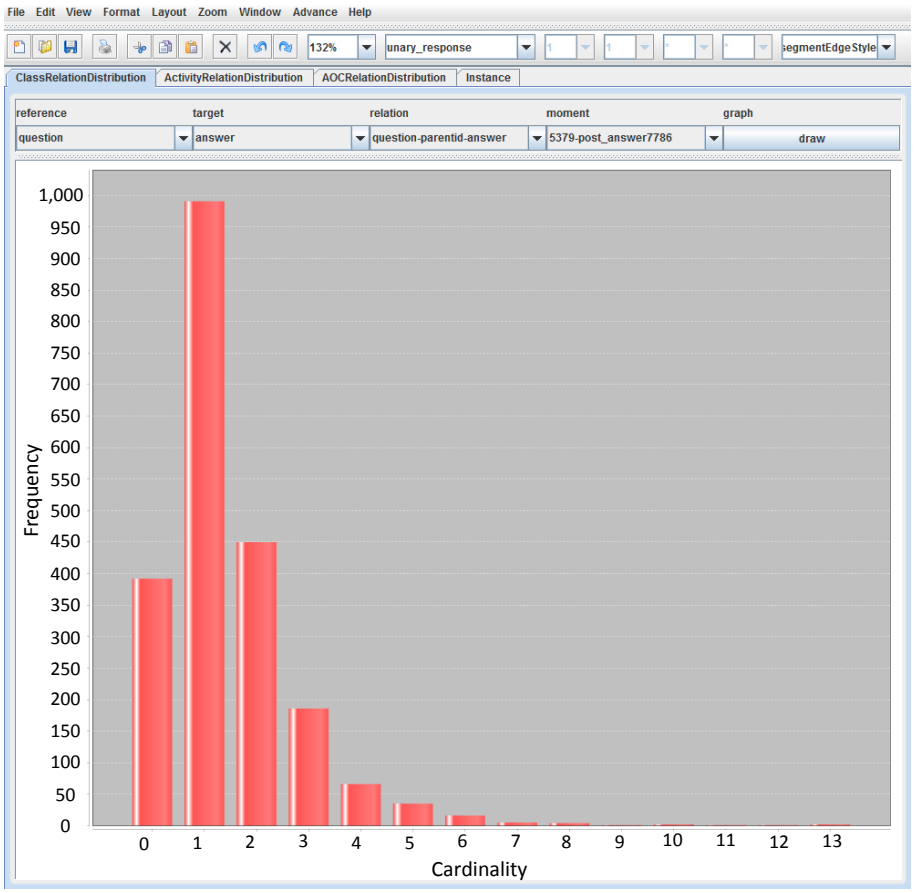


Figure 9.7: This diagram shows the distribution of the cardinalities $\square*$ of $r6$, taking question as a reference. It indicates how many answers a question gets. The bar corresponding to the value "1" means that there are almost 1000 questions that received one answer.

the distribution corresponds to the cardinality constraint $\square*$ at the "answer" side on $r6$. In other words, the distribution presents the relation between the number of answers and questions. As the object model evolves over time, accordingly the distribution changes in different object models. The value in the "moment"

drop-down menu indicates the time (represented by an event) for the presented distribution. Each bar has a cardinality number beneath it, indicating the frequency of questions that have a particular number of answers. By looking at the first three bars, one can see that approx 400+1000+450=1850 questions have up to 2 answers.

The discovered behavioral constraints between activities in Figure 9.6 specify the temporal restrictions on the behavioral perspective. For example, the "unary-precedence" constraint between "register" and "post_question" activities indicates that someone is only able to post questions in the website after registering himself as a user. All these behavioral constraints describe the control-flow in a declarative manner. More precisely, after being registered, a user can post a question, and only then the question can be answered, commented or voted. Similarly, the provided answer can also be commented or voted. Besides, it is possible to get a badge based on one's activities.

In addition to discovering all behavioral constraints to describe the control-flow, our approach can also provide the cardinality numbers distribution of precedence/response target events in terms of a correlation pattern, by using the panel in Figure 9.8. The reference, target and intermediary drop-down menus in the panel indicate the reference activity, target activity and intermediary of the correlation pattern, respectively. In Figure 9.8 the reference activity is "register", the target activity is "post_question" and the intermediary is "user-owneruserid-question" (corresponding to the class relation $r1$). By default, the distribution is for all reference events, in this case for all "register" events. It is possible to inspect the distribution for a particular reference event by setting specific instance values on the drop-down menu displayed in Figure 9.8.

The discovered AOC relationships between activities and classes describe the constraints between events and objects. Here, all the discovered cardinalities on the relationships are $\square 1$ and 1, which indicate the one-to-one relation between events and objects. Thus, in the example, one "register" event corresponds to a "user" object and vice versa.

9.5 OCBC Conformance Checking

The OCBC model discovered in Section 9.4 describes how the Stack Exchange website is operated. However, the real actions may not be entirely consistent with the expected scenario. In this section, we repair the discovered model based on domain knowledge, such that it can be used as a reference model to check conformance.

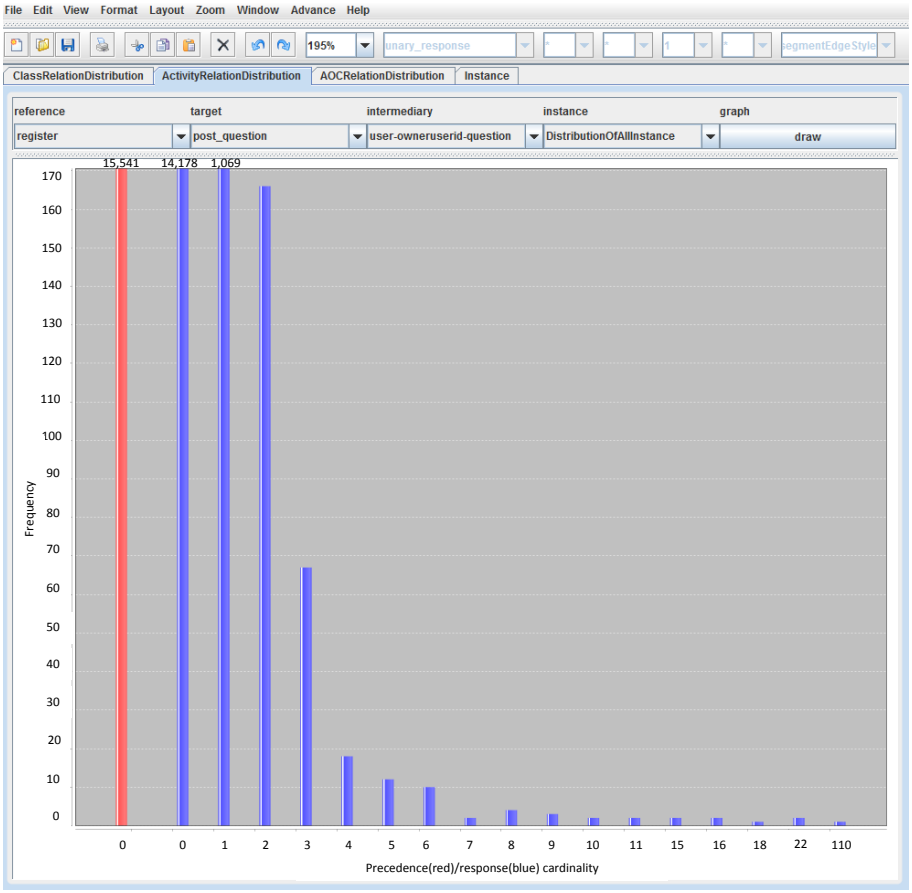


Figure 9.8: This diagram shows the distribution of the behavioral constraint between "register" and "post_question" activities. The red bars correspond to the precedence cardinalities. For instance, the first red bar means that there are 15,541 "register" events that have 0 "post_question" event before. The blue bars correspond to the response cardinalities. For instance, the second blue bar means that there are 1,069 "register" events and each of them has 1 "post_question" event after.

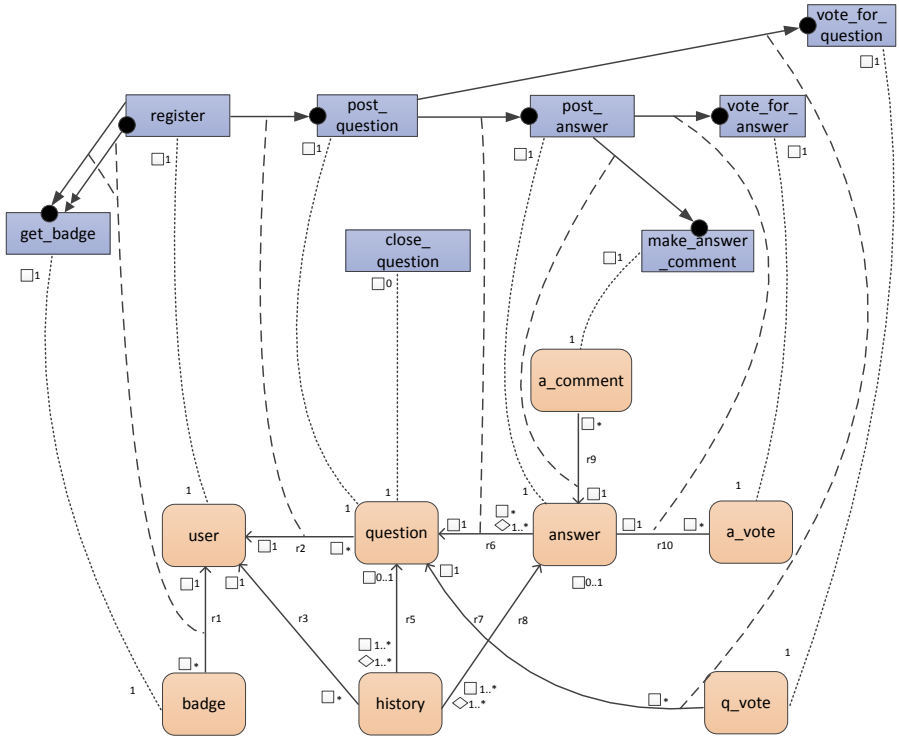


Figure 9.9: The discovered model is repaired to serve as the reference model for conformance checking. We add an eventually cardinality $\Diamond 1..*$ to $r6$, a behavioral constraint between "get_badge" and "register", and an AOC relationship between "close_question" and "question".

In the discovery process, we configured the requirement for fitness as 1, resulting in a model that totally fits the event log. However, the discovered model is not precise in some parts, allowing some unexpected actions to be executed. Based on assumptions derived according to our understanding of the website, some constraints of the discovered model in Figure 9.6 are strengthened, resulting in the reference model depicted in Figure 9.9. The assumptions and strengthened constraints are explained as follows:

- In a normal situation, we require that each question eventually has at least

one corresponding answer, indicated by $\Diamond 1..*$ of $r6$ in Figure 9.9. This requirement is reasonable since the motivation of the website is to post questions and get answers. In other words, we consider the questions without answers as deviations.

- A posted question can be closed if the question is duplicated or of bad quality. The questions that are closed eventually harm the development of the Stack Exchange website, and the website wants to avoid this situation. Based on this knowledge, a question is considered as a deviation if it has a corresponding "close_question" event. Therefore, we add an AOC relationship between the activity "close_question" and the class "question"

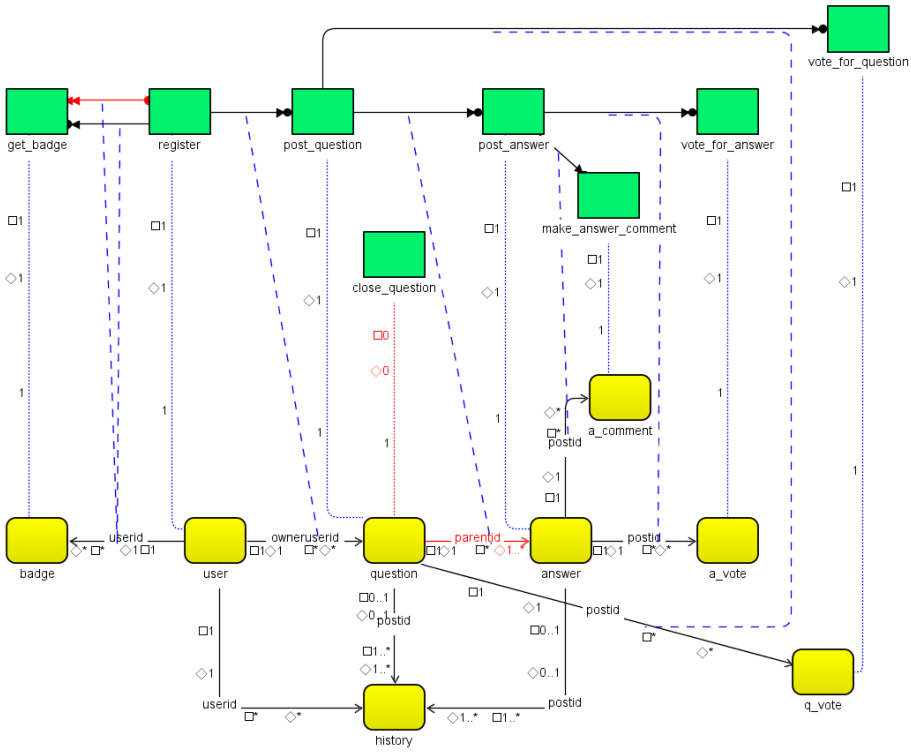


Figure 9.10: Conformance checking results projected onto the OCBC model.

with a cardinality □0 on the activity side in Figure 9.9. In this way, we also test if we can check the conformance in terms of negative constraints.

- The users registered in the website are expected to get badges by being active in posting questions, answers or comments. Thus we add a "response" constraint between "register" and "get_badge" activities in Figure 9.9.

In order to improve the readability of the reference model, we remove the class "q_comment", the activity "make_question_comment" and the involved constraints. Taking the OCBC model in Figure 9.9 and XOC logs extracted in Section 9.3 as input, we check the conformance between them and detect deviations (cf. Chapter 7). The diagnosis result is presented in three views: the model view shown in Figure 9.10, the type view shown in Figure 9.11, and the log view shown in Figure 9.12. In general, the model view provides a helicopter view of the conformance checking result, the type view lists deviations (grouped by types), and the log view supports inspecting deviating objects.

Deviations with respect to the behavioral perspective. In the model view, the "response" constraint between "register" and "get_badge" is highlighted in red,

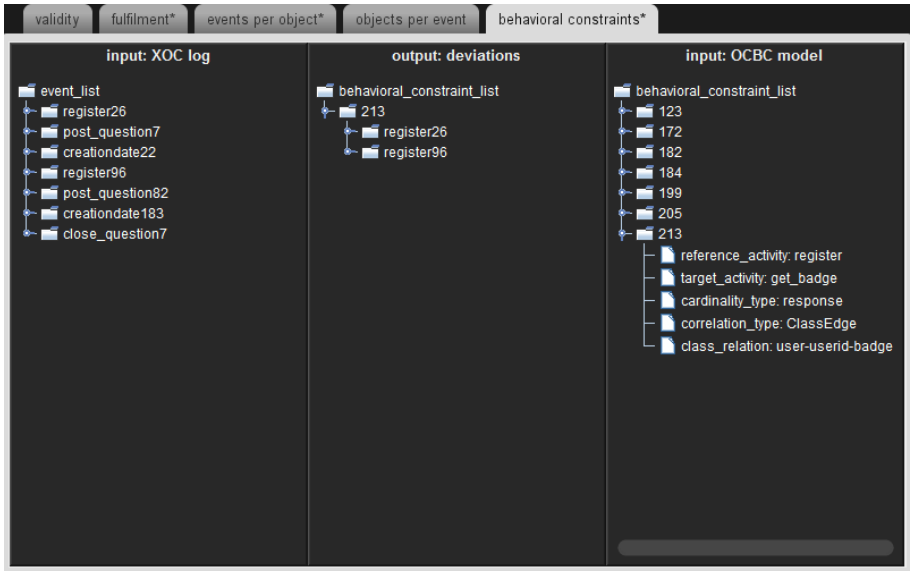


Figure 9.11: Conformance checking results showing different types of deviations.

which indicates that some “register” events are violating this constraint. In the type view, the violated constraints and corresponding deviating events are listed in the “output:deviations” panel. The details of the events and constraints are presented by the “input: XOC log” and “input: OCBC model” panels, respectively. For instance, events “register25” and “register96” are listed, which violate the constraint with an id of “213”. By checking the information in the right panel, the constraint is a “response” constraint between “register” and “get_badge”.

Deviations with respect to the data perspective. In the model view, the $\Diamond 1..*$ on the class relation between “question” and “answer” classes is highlighted in red, which indicates that some “question” objects eventually have no corresponding “answer” objects. All the deviating “question” objects are listed in the type view after clicking the “fulfillment” tab. The log view supports inspecting the deviating objects and their corresponding objects at each moment, i.e., in each

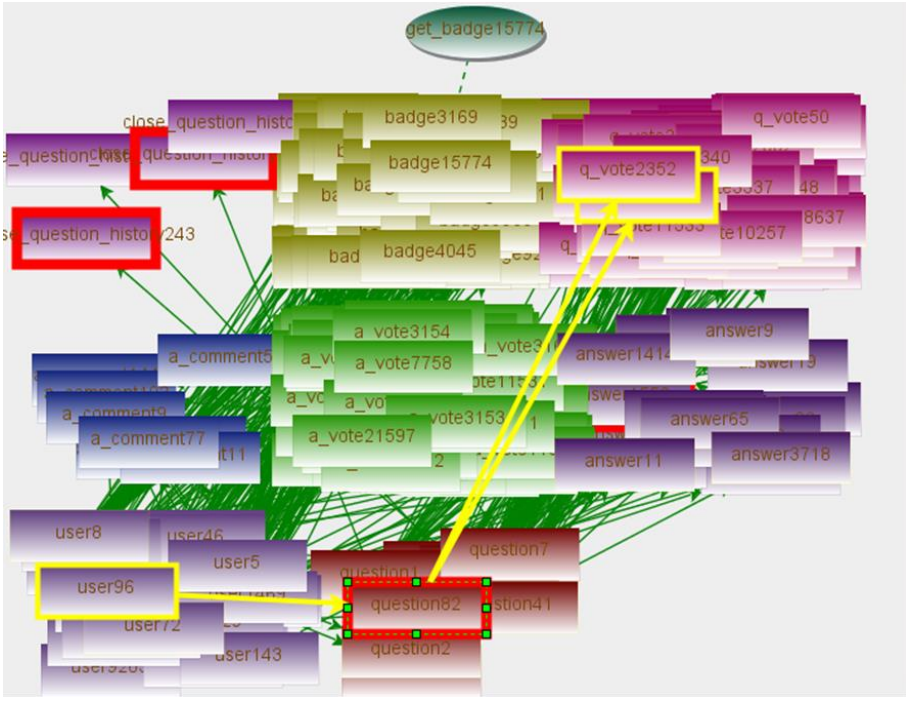


Figure 9.12: Conformance checking results projected onto the XOC log.

object model. Figure 9.12 shows the object model corresponding to the event "get_badge15774". The deviating object "question82" is highlighted in red, since it has no related "answer" objects. Indicated by the yellow lines, it only has one related "user" object and two related "vote" objects.

Deviations related to the interactions between two perspectives. In the model view, $\square 0$ and $\Diamond 0$ on the AOC relation between "close_question" and "question" are highlighted in red, which indicates that some "question" objects are violating the AOC relationship. In the type view, we can use the "events per object" tab to inspect the deviating "question" objects. For instance, "question7" is listed in the "output:deviations" panel since it has a corresponding "close_question" event, which violates the rule that a question should always and eventually has 0 corresponding "close_question" event (indicated by $\square 0$ and $\Diamond 0$).

9.6 OCBC Performance Analysis

Next, we analyze the performance based on the XOC log extracted in Section 9.3 and the OCBC model in Figure 9.6.

In our approach, the performance analysis of the whole model is split into several independent views based on selected correlation patterns (cf. Chapter 8). Therefore, we first extract correlation patterns from the OCBC model in Figure 9.6 (using function *extP* in Chapter 6), resulting in the fourteen patterns presented in Table 9.6. For example, P1 is an extracted pattern, in which "register" is the reference activity, "get_badge" is the target activity and the class relationship r1 serves as the intermediary to connect these two activities. For each pattern, some instances can be derived from the XOC log, and each instance contains a reference event and a set of target events correlated to the reference event by the pattern.

In real applications, it is not necessary to analyze performance for all patterns if the most relevant ones are already known. In the case study of Stack Exchange, the pattern "P5" is one of the most interesting patterns based on common knowledge. For "P5", "post_question" is the reference activity and "post_answer" is the target activity. These two activities are the most important activities in Stack Exchange, since the motivation of the website is providing answers to questions posted by users. Therefore, pattern "P5" appears to be the obvious choice to serve as an example to present our performance analysis result.

Figure 9.13 employs the dotted chart to present the performance analysis result for the pattern "P5". The Y axis indicates that there are almost 2,150 instances while the X axis indicates that events in these instances happened from

Pattern	Reference activity	Target activity	Intermediary
P1	register	get_badge	r1
P2	get_badge	register	r1
P3	register	post_question	r2
P4	post_question	register	r2
P5	post_question	post_answer	r6
P6	post_answer	post_question	r6
P7	post_question	make_question_comment	r4
P8	make_question_comment	post_question	r4
P9	post_question	vote_for_question	r7
P10	vote_for_question	post_question	r7
P11	post_answer	make_answer_comment	r9
P12	make_answer_comment	post_answer	r9
P13	post_answer	vote_for_answer	r10
P14	vote_for_answer	post_answer	r10

Table 9.6: Correlation patterns extracted from the OCBC model in Figure 9.6.

August, 2016 to September, 2018. Each instance corresponds to a row in the dotted chart, which contains a "post_question" event, represented by the green dot, and a set of "post_answer" events, represented by the red dots. By using the results presented by the dotted chart, we can infer some insights explained as follows:

- Questions may have long life cycles. In other words, answers are still received even a long time after the original question creation. This is indicated by the red dots which have long distances from their corresponding green dots.
- The instances are sorted by timestamp. More precisely, the instances at the top of the dotted chart contain questions and answers posted at the moment the website was founded, while the instances at the bottom of the dotted chart contain questions and answers posted more recently. One can see that the density of the red dots is getting high from top to bottom. It indicates that the website is becoming more popular and active, since questions are answered more frequently.
- There is an explosion of questions and answers just after the website was founded-right after the website release (from August, 2016 to September, 2016), a quite steep green line curve and dense red dots can be noticed.

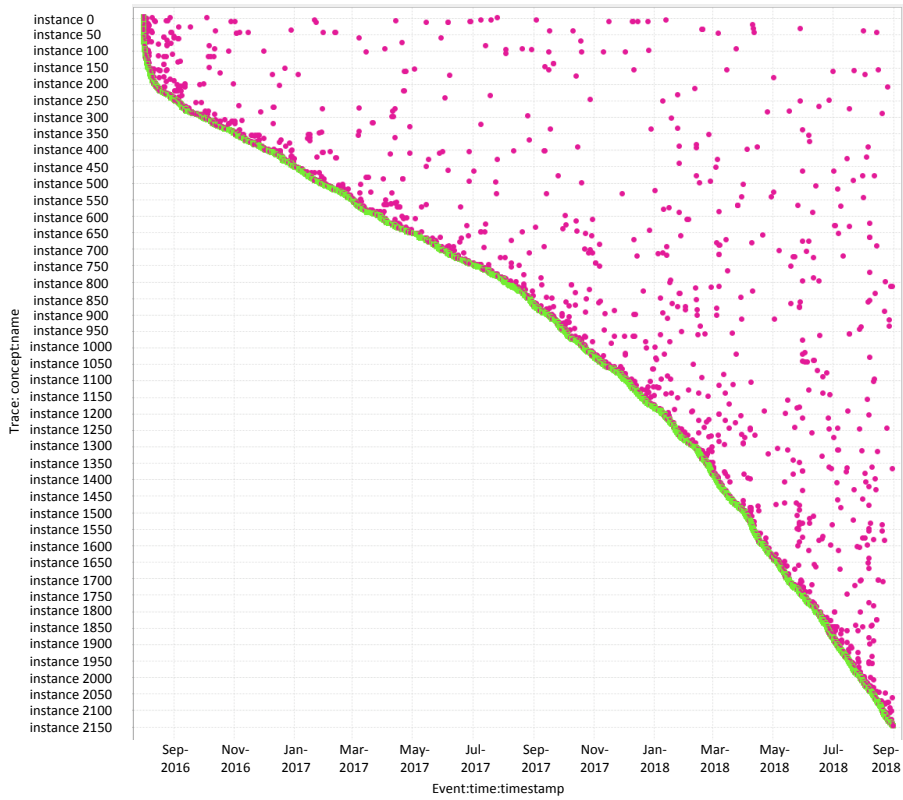


Figure 9.13: The performance analysis result corresponding to pattern “P5”, presented by the dotted chart in the absolute time.

Then the curve goes gently from September, 2016 and becomes steeper in the latest half year (from March, 2018 to September, 2018). A reasonable explanation for this curve behavior is that at the beginning, the website invited some professional people to post questions and provide answers. When the website started to get on the correct track, these invited people stopped posting, leading to fewer questions and answers. In the latest half year, the website became more popular, leading to more questions and answers.

Figure 9.14: The performance analysis result corresponding to pattern “P5”, presented by the dotted chart in the relative time.

Figure 9.13 is based on the absolute time to present the performance result. It is possible to view the result from other angles, such as the one shown in Figure 9.14. This chart provides the performance result in the relative time, by aligning all reference events, i.e., “post_question” events, with the value 0 of the relative time. According to the aligned dotted chart, we can derive some more insights explained as follows:

- All “post_answer” events happen after corresponding “post_question” events- in each row all red dots are placed after the green dot.
- Most answers are provided in a short time period (less than one day) after

the corresponding questions are posted-the dense red dots just after the green dots indicate this.

In addition to dotted charts, our approach can also provide some statistics/-metrics, to reflect the performance. For example, we can consider the time period from the moment that a question is posted to the moment that the last answer corresponding to the question is given as the "question duration" of the instance which contains the question. In terms of the "question duration", the instance $\langle post\_question109, post\_answer7587 \rangle$ has the longest duration: 743 days, 4 hours, 16 minutes and 1 second. The event "post_question109" posted a question with title "Can rule induction be considered a way to hybridize probabilistic/statistical approaches and symbolic approaches?" on August 2rd, 2016. The question was answered on August 15th, 2018, corresponding to the event "post_answer7587". By searching the question online and investigating why the question has a long life cycle, we found that the question was edited on April 14, 2018. A possible explanation is that the modification of the question improves the quality and "revive" the question.

Interestingly, we found the shortest duration is 0. For example, the instance $\langle post\_question1429, post\_answer1430 \rangle$ has a duration of 0. The event "post_question1429" posted the question "How machine learning can help with sustainable development and biological conservation?" on August 7th, 2016. It was answered (corresponding to the event "post_answer1430") at the same time. By searching the question online and investigating why the question was answer without any delay, we found that the question and answer were given by the same user. Actually, the question was a scheduled question rather than a realistic question. The user already knew the answer and he wanted to post some knowledge in the website by this question.

Metric	Average	Deviation	Maximum	Minimum
question duration	43d6h8m17s	-	743d4h16m1s	0s
answer duration	26d8h18m40s	-	738d3h48m54s	0s
length	2.45	1.33	14	1

Table 9.7: Values of some example metrics.

In addition to the metric related to a single instance, there are other metrics that can be used to summarize the performance on a higher level. For instance, the average value for the instance duration is 43 days, 6 hours, 8 minutes and 17 seconds. It indicates that a question is valid or alive (i.e., it can get answers)

for 43 days on average after it is posted. Table 9.7 shows more metrics to reflect the performance, where “answer duration” means the time period between the first answer and the last answer while the “length” means the number of events in an instance.

9.7 Summary

In this chapter, we discussed a case study based on a real-life data set from the Stack Exchange website. Based on domain knowledge about data schema and activities, we parsed the data set and extracted XOC logs, which were taken as input to implement OCBC techniques.

Using discovery technique, an OCBC was discovered to describe how the Question & Answer process is operated in the website. By repairing some constraints in the discovered model and considering it as the reference model, conformance was checked to detect deviations, such as questions without any answers. At last, performance was analyzed for an example pattern involved “post_question” and “post_answer” activities, which provided some insights on the time perspective, e.g., most answers are given in one day after the corresponding questions are posted.

The case study shows that the OCBC techniques can be applied to real-life data and provide novel insights.

Chapter 10

Conclusions and Future Work

This chapter concludes the thesis. In Section 10.1 the main contributions of this thesis are summarized. Section 10.2 discusses the limitations and open problems. Finally, we propose some ideas for future work in Section 10.3.

10.1 Contributions of the Thesis

In this thesis, we proposed a range of techniques for analyzing business processes in artifact-centric information systems. These techniques are all based on Object-Centric Behavioral Constraint (OCBC) models. In general, our work covers most types of process mining, i.e., extracting event logs from execution data, discovering models from event logs, and checking conformance and analyzing performance based on logs and models.

The XOC log format to organize data and the OCBC models to describe business processes form the cornerstones of our research. Several novel techniques were created to derive insights, which are summarized as follows:

XOC event logs and extraction. Chapter 3 defined the notion of object-centric event data to abstract the data generated by artifact-centric information systems. Such data are different from case-centric event data and have their own features. In order to organize such data, Chapter 4 introduced the XOC log format which does not require the case notion. Besides, an approach was proposed to automatically extract XOC logs from object-centric event data.

OCBC models and discovery. Chapter 5 illustrated the OCBC models by explaining all the involved elements: the data perspective, e.g., classes; the

behavioral perspective, e.g., activities; and then the AOC relationships combining the first two. Chapter 6 presented approaches to automatically discover OCBC models from XOC logs. First, a basic approach was illustrated to discover models from clean logs. Then, a more robust discovery approach was proposed to deal with noise in real life data.

OCBC conformance checking. Based on an XOC log and a manually designed or discovered OCBC model, Chapter 7 showed techniques to diagnose the conformance between them. By taking the data perspective into consideration, it is possible to detect and diagnose a range of conformance problems that would have remained undetected by conventional approaches. The diagnostic results (i.e., deviations related to the behavioral perspective, data perspective and interactions) were present in three different views: rule view, log view, and model view. Besides, metrics such as fitness, precision and generalization were defined to quantify the degree of conformance.

OCBC performance analysis. Chapter 8 analyzed the performance in terms of frequencies and times by taking an XOC log and an OCBC model as input. In order to show performance results from different angles, (dotted and column) charts, and indicators were used. With the obtained results in hands, it was possible to map them onto the model, and from this step important bottlenecks could be revealed.

Case study using real life data. To link all techniques in a consistent manner, in Chapter 9 a case study showed how OCBC techniques can be applied to real-life data.

10.2 Limitations

The thesis provides a range of OCBC/XOC-based techniques supporting the different types of process mining. Despite the broad coverage of process mining there are several known limitations. In this section, these limitations are discussed making clear that by addressing them the OCBC techniques would become more stable and complete.

Domain knowledge. In Chapter 4, we presented how XOC logs can be extracted from database tables and change logs (such as redo logs and change tables). The events can be derived from database changes recorded in the change logs, and this is done by using domain knowledge to identify the activities of events. Besides, it is possible to extract events from timestamp columns in database tables. In this situation, the domain knowledge is also required when identifying

the activities of events, e.g., the events corresponding to the "creation_date" column in the "order" table are "create order" events. In summary, and as a rule of thumb, the domain knowledge must be used and considered as a direct dependency when identifying activities of events during the log extraction. For instance, we can consider column names as activities, which are not intuitive and risky (since different tables can have the same column name).

Model Complexity. An OCBC model consists of activities, classes and various constraints. However, as the number of activities and classes increases, the complexity of the corresponding model can rise, decreasing its readability. OCBC models tend to be larger and more complex, since they include both behavioral and data perspectives (while most other models only include the behavioral perspective), which increases the model complexity.

Large scale data. The input data for OCBC techniques from databases (of artifact-centric information systems) can be very large. As a result, the current OCBC techniques still need to be improved to deal with the big data topic. In its current state, the performance analysis is time-consuming when the model is complex and there is no knowledge to identify some interesting correlation patterns, i.e., we have to compute the performance for all patterns.

A website providing various OCBC software. In order to apply OCBC techniques, we built a website (https://www.win.tue.nl/ocbc/) to present the involved software in this thesis. Moreover, this website also offers manuals to introduce software and example data for experiments.

10.3 Future Work

This final section sketches ideas for further research in OCBC techniques that extend beyond the scope of this thesis.

Attribute level. In Chapter 5, the semantics of constraints in OCBC models are defined on the entity level. For instance, an AOC relationship between the activity "create order" and the class "order" specifies the correspondence between "create order" events and "order" objects. It is possible to intensify the semantics by incorporating the attribute level. As a result, the AOC relationship can also indicate the correspondence between "create order" event attributes and "order" object attributes. In Chapter 7, the conformance diagnosis reveals the deviations about the instances on the entities level, i.e., events and objects. Similarly, it is also possible to check conformance on the attribute level, i.e., taking into

consideration the attribute values.

Semantics and reasoning on OCBC models. Crucial reasoning tasks that exist for Declare and the like (such as consistency, dead activities, etc.) are not discussed in the context of OCBC models in this thesis, which may lead to potential problems. For instance, when dealing with noise, it is well known that as soon as constraints with support less than 100% are retained, the algorithm could produce a final inconsistent model [38]. In future, these tasks should be taken into consideration to make our techniques more robust.

Model patterns. It is possible to identify typical behavioral *patterns* that involve multiple instances or interaction between structure and behavior. Along this line, we plan to study the effect of introducing *subtyping* in the data model, a constraint present in all data modeling approaches. The interplay between behavioral constraints and subtyping gives rise to other interesting behavioral patterns. For example, *implicit choices* may be introduced through subtyping. Consider a response constraint pointing to a *payment* class with two subclasses *credit card payment* and *cash payment*. Whenever the response constraint is activated and a payment is expected, such an obligation can be fulfilled by either paying via cash or credit card.

Distributed processing. It is promising to relate process mining to Big Data technologies. In order to solve the limitations of OCBC techniques when dealing with large scale data, we can investigate how to incorporate the parallel computing approaches into these techniques. For instance, since OCBC models correlate events and analyze performance based on individual correlation patterns rather than the whole process, the performance analysis on the whole model can be split into independent smaller tasks for all correlation patterns. Based on this idea, parallel computing approaches can be employed to execute the independent tasks and integrate the results onto the whole OCBC model. Similarly, the parallel manner can be also applied to log extraction and model discovery.

Bibliography

[1] IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std. 1849-2016*, pages i–48, 2016. (Cited on pages 50 and 87.)

[2] Arya Adriansyah. *Performance analysis of business processes from event logs and given process models*. PhD thesis, Master Thesis. Eindhoven University of Technology, 2009. (Cited on pages 272 and 314.)

[3] Arya Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2014. (Cited on pages 235 and 267.)

[4] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment based precision checking. In *International Conference on Business Process Management*, pages 137–149. Springer, 2012. (Cited on page 267.)

[5] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Measuring precision of modeled behavior. *Information systems and e-Business Management*, 13(1):37–67, 2015. (Cited on pages 235, 246, and 267.)

[6] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Towards robust conformance checking. In *International Conference on Business Process Management*, pages 122–133. Springer, 2010. (Cited on pages 235 and 267.)

[7] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 55–64. IEEE, 2011. (Cited on pages 224, 235, and 267.)

[8] Ruth S. Aguilar-Saven. Business process modelling: Review and framework. *International Journal of production economics*, 90(2):129–149, 2004. (Cited on page 32.)

[9] Alessandro Artale, Roman Kontchakov, Vladislav Ryzhikov, and Michael Zakharyaschev. A cookbook for temporal conceptual data modelling with description logics. *ACM Transactions on Computational Logic (TOCL)*, 15(3):25, 2014. (Cited on page 124.)

[10] Jana Bauckmann, Ulf Leser, Felix Naumann, and Véronique Tietz. Efficiently detecting inclusion dependencies. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1448–1450. IEEE, 2007. (Cited on page 219.)

[11] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on UML class diagrams. *Artificial intelligence*, 168(1-2):70–118, 2005. (Cited on pages 28 and 166.)

[12] Robin Bergenthum, Jörg Desel, Robert Lorenz, and Sebastian Mauser. Process mining based on regions of languages. In *International Conference on Business Process Management*, pages 375–383. Springer, 2007. (Cited on page 216.)

[13] Alessandro Berti and Wil M. P. van der Aalst. Starstar models: Process analysis on top of databases. *arXiv preprint arXiv:1811.08143*, 2018. (Cited on page 118.)

[14] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *International Conference on Business Process Management*, pages 288–304. Springer, 2007. (Cited on page 41.)

[15] Kamal Bhattacharya, Richard Hull, and Jianwen Su. A data-centric design methodology for business processes. In *Handbook of Research on Business Process Modeling*, pages 503–531. IGI Global, 2009. (Cited on page 47.)

[16] Grady Booch. *The Unified Modeling Language User Guide*. Pearson Education India, 2005. (Cited on pages 28 and 122.)

[17] Joos C. A. M. Buijs. Flexible evolutionary algorithms for mining structured process models. *Unpublished Ph. D. Thesis, Eindhoven University of Technology, Netherland*, 2014. (Cited on page 220.)

[18] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 23(01):1440001, 2014. (Cited on pages 216 and 252.)

[19] Forrest Burnson. The buffet of ERP modules: Why you should build your system. `https://www.softwareadvice.com/resources/erp-modules-benefits`. (Cited on page 12.)

[20] Diego Calvanese, Tahir E. Kalayci, Marco Montali, and Stefano Tinella. Ontology-based data access for extracting event logs from legacy data: The onprom tool and methodology. In *BIS 2017*, pages 220–236. Springer, 2017. (Cited on page 119.)

[21] Diego Calvanese, Marco Montali, Alifah Syamsiyah, and Wil M. P. van der Aalst. Ontology-driven extraction of event logs from relational databases. In *BPM 2015 workshops*, pages 140–153. Springer, 2015. (Cited on page 119.)

[22] Jorge Cardoso, Robert P. Bostrom, and Amit Sheth. Workflow management systems and ERP systems: Differences, commonalities, and applications. *Information Technology and Management*, 5(3-4):319–338, 2004. (Cited on pages 8 and 12.)

[23] Fabio Casati, Malu Castellanos, Umeshwar Dayal, and Norman Salazar. A generic solution for warehousing business process data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1128–1137. VLDB Endowment, 2007. (Cited on page 8.)

[24] Tian Chao, David Cohn, Adrian Flatgard, Sandy Hahn, Mark Linehan, Prabir Nandi, Anil Nigam, Florian Pinel, John Vergo, and Frederick y Wu. Artifact-based transformation of IBM global financing. In *International Conference on Business Process Management*, pages 261–277. Springer, 2009. (Cited on page 44.)

[25] Peter P. S. Chen. The entity-relationship model-toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976. (Cited on pages 26, 55, 122, 128, and 162.)

[26] Carolina Ming Chiao, Vera Kunzle, and Manfred Reichert. Integrated modeling of process-and data-centric software systems with PHILharmonicFlows. In *Communicating Business Process and Software Models Quality, Understandability, and Maintainability (CPSM), 2013 IEEE 1st International Workshop on*, pages 1–10. IEEE, 2013. (Cited on page 162.)

[27] Edgar F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970. (Cited on page 30.)

[28] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. *IEEE Data Eng. Bull.*, 32(3):3–9, 2009. (Cited on pages 41, 44, 47, 123, and 163.)

[29] Stack Exchange Community. Stack exchange data dump. `https://archive.org/details/stackexchange`. 2019. (Cited on pages 318 and 321.)

[30] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. BPMN miner: Automated discovery of BPMN process models with hierarchical structure. *Information Systems*, 56:284–303, 2016. (Cited on pages 214 and 220.)

[31] Massimiliano de Leoni, Fabrizio M. Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *International Conference on Business Process Management*, pages 82–97. Springer, 2012. (Cited on page 268.)

[32] Massimiliano de Leoni, Fabrizio M. Maggi, and Wil M. P. van der Aalst. An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data. *Information Systems*, 47:258–277, 2015. (Cited on page 268.)

[33] Massimiliano de Leoni and Wil M. P. van der Aalst. Data-aware process mining: discovering decisions in processes using alignments. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 1454–1461. ACM, 2013. (Cited on pages 124, 162, 213, and 217.)

[34] Massimiliano de Leoni, Wil M. P. van der Aalst, and Boudewijn F. van Dongen. Data-and resource-aware conformance checking of business processes. In *International Conference on Business Information Systems*, pages 48–59. Springer, 2012. (Cited on pages 225, 233, 234, and 268.)

[35] Ana K. A. de Medeiros, Wil M. P. van der Aalst, and Anton J. M. M. Weijters. Quantifying process equivalence based on observed behavior. *Data & knowledge engineering*, 64(1):55–74, 2008. (Cited on pages 235 and 267.)

[36] Ana K. A. de Medeiros, Anton J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007. (Cited on page 216.)

[37] Søren Debois, Thomas T. Hildebrandt, Paw H. Laursen, and Kenneth R. Ulrik. Declarative process mining for DCR graphs. In *Proceedings of the Symposium on Applied Computing*, pages 759–764. ACM, 2017. (Cited on page 218.)

[38] Claudio Di Ciccio, Fabrizio M. Maggi, Marco Montali, and Jan Mendling. Resolving inconsistencies and redundancies in declarative process models. *Information Systems*, 64:425–446, 2017. (Cited on page 344.)

[39] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A. Reijers, et al. *Fundamentals of business process management*, volume 1. Springer, 2013. (Cited on pages 8, 10, and 12.)

[40] Marlon Dumas, Wil M. P. van der Aalst, and Arthur H. M. ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005. (Cited on page 8.)

[41] Ramez Elmasri. *Fundamentals of database systems*. Pearson Education India, 2008. (Cited on pages 27 and 29.)

[42] Hans-Erik Eriksson and Magnus Penker. Business modeling with UML. *New York*, pages 1–12, 2000. (Cited on pages 28 and 55.)

[43] Dirk Fahland, Massimiliano de Leoni, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Behavioral conformance of artifact-centric process models. In *International Conference on Business Information Systems*, pages 37–49. Springer, 2011. (Cited on pages 225, 233, and 268.)

[44] Dirk Fahland, Massimiliano de Leoni, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking of interacting processes with overlapping instances. In *International Conference on Business Process Management*, pages 345–361. Springer, 2011. (Cited on page 268.)

[45] Wojciech Fliegner. Extracting process-related information from ERP systems for process discovery. *Research in Logistics & Production*, 4, 2014. (Cited on page 48.)

[46] Hartmann J. Genrich. Predicate/transition nets. In *Advances in Petri Nets 1986, Part I on Petri Nets: Central Models and Their Properties*, pages 207–247, Berlin, Heidelberg, 1987. Springer-Verlag. (Cited on page 162.)

[47] Hartmann J. Genrich and Kurt Lautenbach. The analysis of distributed systems by means of predicate/transition-nets. In *Semantics of Concurrent Computation*, pages 123–146. Springer, 1979. (Cited on page 162.)

[48] Javier Gonzalez-Huerta, Anis Boubaker, and Hafedh Mili. A business process re-engineering approach to transform bpmn models to software artifacts. In *International Conference on E-Technologies*, pages 170–184. Springer, 2017. (Cited on page 122.)

[49] Eduardo González López de Murillas. *Process mining on databases: Extracting event data from real life data sources*. PhD thesis, Eindhoven University of Technology, 2019. (Cited on page 71.)

[50] Eduardo González López de Murillas, Hajo A. Reijers, and Wil M. P. van der Aalst. Connecting databases with process mining: A meta model and toolset. In *Enterprise, Business-Process and Information Systems Modeling*, pages 231–249. Springer, 2016. (Cited on page 118.)

[51] Eduardo González López de Murillas, Wil M. P. van der Aalst, and Hajo A. Reijers. Process mining on databases: Unearthing historical data from redo logs. In *International Conference on Business Process Management*, pages 367–385. Springer, 2015. (Cited on pages 71, 76, 78, and 119.)

[52] Heidi Gregersen and Christian S. Jensen. Temporal entity-relationship models-a survey. *IEEE Transactions on knowledge and data engineering*, 11(3):464–497, 1999. (Cited on page 124.)

[53] Object Management Group. *OMG Unified Modeling Language 2.5*. OMG, 2013. (Cited on pages 28, 128, and 162.)

[54] Object Management Group. *Object Constraint Language, Version 2.4*. OMG, http://www.omg.org/spec/OCL/2.4/, 2014. (Cited on page 132.)

[55] Object Management Group et al. Business process model and notation (BPMN): version 2.0. *Object Management Group*, 2011. (Cited on pages 35 and 122.)

[56] Christian W. Günther. *Process mining in flexible environments*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2009. (Cited on pages 272 and 313.)

[57] Christian W. Günther and Wil M. P. van der Aalst. A generic import framework for process event logs. In *International Conference on Business Process Management*, pages 81–92. Springer, 2006. (Cited on page 118.)

[58] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007. (Cited on page 216.)

[59] Rania B. Halima, Imen Zouaghi, Slim Kallel, Walid Gaaloul, and Mohamed Jmaiel. Formal verification of temporal constraints and allocated cloud resources in business processes. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, pages 952–959. IEEE, 2018. (Cited on page 268.)

[60] Terry Halpin and Tony Morgan. *Information Modeling and Relational Databases*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. (Cited on pages 129 and 162.)

[61] Michael Hammer. What is business process management? In *Handbook on business process management 1*, pages 3–16. Springer, 2015. (Cited on page 272.)

[62] Thomas T. Hildebrandt and Raghava R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. *arXiv preprint arXiv:1110.4161*, 2011. (Cited on page 218.)

[63] Bart F. A. Hompes, Joos C. A. M. Buijs, and Wil M. P. van der Aalst. A generic framework for context-aware process performance analysis. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 300–317. Springer, 2016. (Cited on page 314.)

[64] Peter T. G. Hornix. Performance analysis of business processes through process mining. *Master's Thesis, Eindhoven University of Technology*, 2007. (Cited on pages 272, 313, and 315.)

[65] Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 1152–1163. Springer, 2008. (Cited on pages 41 and 47.)

[66] Richard Hull, Elio Damaggio, Riccardo De Masellis, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath III, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, et al. Business artifacts with guard-stage-milestone lifecycles: Managing artifact interactions with conditions and events. In *Proceedings of the 5th ACM international conference on Distributed event-based system*, pages 51–62. ACM, 2011. (Cited on pages 44, 123, and 163.)

[67] Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Terry Heath, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Piyawadee Sukaviriya, et al. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In *International Workshop on Web Services and Formal Methods*, pages 1–24. Springer, 2010. (Cited on pages 44 and 163.)

[68] Jon E. Ingvaldsen and Jon A. Gulla. Preprocessing support for large scale process mining of SAP transactions. In *BPM 2007 workshops*, pages 30–41. Springer, 2007. (Cited on page 119.)

[69] Stefan Jablonski and Christoph Bussler. *Workflow management: modeling concepts, architecture and implementation*, volume 392. International Thomson Computer Press London, 1996. (Cited on page 8.)

[70] Mieke Jans and Pnina Soffer. From relational database to event log: Decisions with quality impact. In *International Workshop on Quality Data for Process Analytics*. Springer, 2017. (Cited on page 119.)

[71] Kurt Jensen. Coloured Petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer-Verlag, Berlin, 1987. (Cited on page 33.)

[72] Kurt Jensen. Coloured Petri nets. In *Advances in Petri Nets 1986, Part I on Petri Nets: Central Models and Their Properties*, pages 248–299, Berlin, Heidelberg, 1987. Springer-Verlag. (Cited on pages 162 and 256.)

[73] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013. (Cited on pages 33 and 124.)

[74] Kurt Jensen and Lars M. Kristensen. *Coloured Petri nets: modelling and validation of concurrent systems*. Springer Science & Business Media, 2009. (Cited on pages 33 and 162.)

[75] Vera Künzle. *Object-aware process management*. PhD thesis, University of Ulm, 2013. (Cited on page 162.)

[76] Vera Künzle and Manfred Reichert. PHILharmonicFlows: towards a framework for object-aware process management. *Journal of Software Maintenance and Evolution: Research and Practice*, 23(4):205–244, 2011. (Cited on page 162.)

[77] Pia Landergren. Net presents ERP vendors with new challenges. `http://www.itworld.com`. 2002. (Cited on page 8.)

[78] Maikel Leemans and Wil M. P. van der Aalst. Discovery of frequent episodes in event logs. In *International Symposium on Data-Driven Process Discovery and Analysis*, pages 1–31. Springer, 2014. (Cited on page 220.)

[79] Sander J. J. Leemans. *Inductive visual Miner*. (Cited on page 311.)

[80] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013. (Cited on pages 216 and 220.)

[81] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013. (Cited on pages 216 and 220.)

[82] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from incomplete event logs. In

International Conference on Applications and Theory of Petri Nets and Concurrency, pages 91–110. Springer, 2014. (Cited on pages 216 and 220.)

[83] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Process and deviation exploration with inductive visual miner. *BPM (Demos)*, 1295:46, 2014. (Cited on page 213.)

[84] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Scalable process discovery with guarantees. In *International Conference on Enterprise, Business-Process and Information Systems Modeling*, pages 85–101. Springer, 2015. (Cited on pages 216 and 220.)

[85] Michael Leyer. Towards a context-aware analysis of business process performance. In *Proceedings of the 15th Pacific Asia Conference of Information Systems*, 2011. (Cited on page 314.)

[86] Guangming Li and Renata M. de Carvalho. Dealing with artifact-centric systems: A process mining approach. In *EMISA*, pages 80–84, 2018. (Cited on page 19.)

[87] Guangming Li, Renata M. de Carvalho, and Wil M. P. van der Aalst. Automatic Discovery of Object-Centric Behavioral Constraint Models. In *International Conference on Business Information Systems*, pages 43–58. Springer, 2017. (Cited on pages 119 and 120.)

[88] Guangming Li, Renata M. de Carvalho, and Wil M. P. van der Aalst. Configurable event correlation for process discovery from object-centric event data. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 203–210. IEEE, 2018. (Cited on page 79.)

[89] Guangming Li, Renata M. de Carvalho, and Wil M. P. van der Aalst. A model-based framework to automatically generate semi-real data for evaluating data analysis techniques. In *International Conference on Enterprise Information Systems*. SCITEPRESS, 2019. (Cited on page 55.)

[90] Guangming Li, Renata M. de Carvalho, and Wil M. P. van der Aalst. Object-centric behavioral constraint models: A hybrid model for behavioral and data perspectives. In *The 34th ACM/SIGAPP Symposium On Applied Computing (SAC 2019)*, pages 48–56. ACM, 2019. (Cited on page 124.)

[91] Guangming Li, Eduardo González López de Murillas, Renata M. de Carvalho, and Wil M. P. van der Aalst. Extracting object-centric event logs to

support process mining on databases. In *Information Systems in the Big Data Era*, pages 182–199. Springer, 2018. (Cited on page 103.)

[92] Niels Lohmann. Compliance by design for artifact-centric business processes. In *Business Process Management (BPM 2011)*, volume 6896 of *lncs*. Springer, 2011. (Cited on pages 123 and 163.)

[93] Xixi Lu, Marijn Nagelkerke, Dennis van de Wiel, and Dirk Fahland. Discovering interacting artifacts from ERP systems. *IEEE Transactions on Services Computing*, 8(6):861–873, 2015. (Cited on pages 88, 119, and 218.)

[94] Fabrizio M. Maggi, Marlon Dumas, Luciano García-Bañuelos, and Marco Montali. Discovering data-aware declarative process models from event logs. In *Business Process Management*, pages 81–96. Springer, 2013. (Cited on pages 38, 39, 214, 217, 218, and 220.)

[95] Fabrizio M. Maggi, Arjan J. Mooij, and Wil M. P. van der Aalst. User-guided discovery of declarative process models. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 192–199. IEEE, 2011. (Cited on pages 39, 214, 217, and 220.)

[96] Fabrizio M. Maggi, Jagadeesh C. Bose R.P., and Wil M. P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In *International Conference on Advanced Information Systems Engineering*, pages 270–285. Springer, 2012. (Cited on pages 217 and 220.)

[97] Felix Mannhardt. *Multi-perspective process mining*. PhD thesis, University of Michigan, Eindhoven, 2018. (Cited on pages 162 and 268.)

[98] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016. (Cited on page 234.)

[99] Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers, and Wil M. P. van der Aalst. Data-driven process discovery-revealing conditional infrequent behavior from event logs. In *International Conference on Advanced Information Systems Engineering*, pages 545–560. Springer, 2017. (Cited on pages 217 and 220.)

[100] Ronny S. Mans, Nick C. Russell, Wil M. P. van der Aalst, Piet J. M. Bakker, Arnold J. Moleman, and Monique W. M. Jaspers. Proclets in healthcare. *Journal of Biomedical Informatics*, 43(4):632–649, 2010. (Cited on page 44.)

[101] Fredrik Milani and Fabrizio M. Maggi. A comparative evaluation of log-based process performance analysis techniques. *arXiv preprint arXiv:1804.03965*, 2018. (Cited on page 313.)

[102] Hafedh Mili et al. Business process modeling languages: Sorting through the alphabet soup. *ACM Computing Surveys (CSUR)*, 43(1):4, 2010. (Cited on page 32.)

[103] Peter Mortensen. Database schema documentation for the public data dump and sede. `https://meta.stackexchange.com/questions/2677/database-schema-documentation-for-the-public-data-dump-and-sede`. 2019. (Cited on page 324.)

[104] Jorge Muñoz-Gama and Josep Carmona. A fresh look at precision in process conformance. In *International Conference on Business Process Management*, pages 211–226. Springer, 2010. (Cited on pages 225 and 267.)

[105] Jorge Munoz-Gama and Josep Carmona. Enhancing precision in process conformance: stability, confidence and severity. In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 184–191. IEEE, 2011. (Cited on page 267.)

[106] Hoang Nguyen, Marlon Dumas, Arthur H. M. ter Hofstede, Marcello La Rosa, and Fabrizio M. Maggi. Business process performance mining with staged process flows. In *International Conference on Advanced Information Systems Engineering*, pages 167–185. Springer, 2016. (Cited on page 314.)

[107] Anil Nigam and Nathan S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003. (Cited on pages 41, 47, 119, 123, and 163.)

[108] Takahide Nogayama and Haruhisa Takahashi. Estimation of average latent waiting and service times of activities from event logs. In *International Conference on Business Process Management*, pages 172–179. Springer, 2015. (Cited on page 315.)

[109] Erik H. Nooijen. Artifact-centric process analysis–process discovery in erp systems. Master's thesis, Master Thesis. Eindhoven University of Technology, 2012. (Cited on page 218.)

[110] Erik H. Nooijen, Boudewijn F. van Dongen, and Dirk Fahland. Automatic discovery of data-centric and artifact-centric processes. In *International Conference on Business Process Management*, pages 316–327. Springer, 2012. (Cited on page 218.)

[111] Ricardo Pérez-Castillo and et al. Assessing event correlation in non-process-aware information systems. *Software & Systems Modeling*, 13(3):1117–1139, 2014. (Cited on page 119.)

[112] Maja Pesic. *Constraint-based workflow management systems: shifting control to users*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2008. (Cited on page 39.)

[113] Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *International conference on business process management*, pages 169–180. Springer, 2006. (Cited on pages 38 and 124.)

[114] Paul Pichler, Barbara Weber, Stefan Zugal, Jakob Pinggera, Jan Mendling, and Hajo A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *International Conference on Business Process Management*, pages 383–394. Springer, 2011. (Cited on page 39.)

[115] David Piessens. Event log extraction from SAP ECC 6.0. *Event Log Extraction from SAP ECC*, 6, 2011. (Cited on page 71.)

[116] David Piessens, Moe T. Wynn, Michael J. Adams, and Boudewijn F. van Dongen. Performance analysis of business process models with advanced constructs. In *21st Australasian Conference on Information Systems (ACIS 2010): Information Systems: Defining and Establishing a High Impact Discipline*, December 2010. (Cited on page 314.)

[117] Viara Popova, Dirk Fahland, and Marlon Dumas. Artifact lifecycle discovery. *International Journal of Cooperative Information Systems*, 24(01):1–44, 2015. (Cited on page 218.)

[118] Lihi Raichelson and Pnina Soffer. Merging event logs with many to many relationships. In *BPM 2014 workshops*, pages 330–341. Springer, 2014. (Cited on page 119.)

[119] Raghu Ramakrishnan and Johannes Gehrke. *Database management systems*. McGraw Hill, 2000. (Cited on pages 26, 29, and 30.)

[120] Mohammad A. Rashid, Liaquat Hossain, and Jon D. Patrick. The evolution of ERP systems: A historical perspective, 2002. (Cited on page 10.)

[121] Pedro H. P. Richetti, Fernanda A. Baião, and Flávia M. Santoro. Declarative process mining: reducing discovered models complexity by pre-processing event logs. In *International Conference on Business Process Management*, pages 400–407. Springer, 2014. (Cited on page 217.)

[122] Marcella Rovani, Fabrizio M. Maggi, Massimiliano de Leoni, and Wil M. P. van der Aalst. Declarative process mining in healthcare. *Expert Systems with Applications*, 42(23):9236–9251, 2015. (Cited on pages 217 and 220.)

[123] Anne Rozinat. *Process mining: conformance and extension*. PhD thesis, Eindhoven University of Technology, Eindhoven, 2010. (Cited on pages 235 and 267.)

[124] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in ProM. In *International Conference on Business Process Management*, pages 420–425. Springer, 2006. (Cited on page 217.)

[125] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008. (Cited on pages 224, 225, 233, 235, and 267.)

[126] João C. Seco, Søren Debois, Thomas Hildebrandt, and Tijs Slaats. Reseda: Declaring live event-driven computations as reactive semi-structured data. In *2018 IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 75–84. IEEE, 2018. (Cited on page 146.)

[127] Natalia Sidorova, Christian Stahl, and Nikola Trčka. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Information Systems*, 36(7):1026–1043, 2011. (Cited on pages 124 and 162.)

[128] Abraham Silberschatz, Henry F. Korth, Shashank Sudarshan, et al. *Database system concepts*, volume 4. McGraw-Hill New York, 1997. (Cited on pages xvi, 26, 27, 28, and 29.)

[129] Yannis Sismanis, Paul Brown, Peter J. Haas, and Berthold Reinwald. GORDIAN: efficient and scalable discovery of composite keys. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 691–702. VLDB Endowment, 2006. (Cited on page 219.)

[130] Minseok Song and Wil M. P. van der Aalst. Supporting process mining by showing events at a glance. In *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145, 2007. (Cited on pages 276 and 313.)

[131] Nikola Trčka, Wil M. P. van der Aalst, and Natalia Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In *International Conference on Advanced Information Systems Engineering*, pages 425–439. Springer, 2009. (Cited on page 162.)

[132] Wil M. P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999. (Cited on page 122.)

[133] Wil M. P. van der Aalst. Process-aware information systems: Lessons to be learned from process mining. In *Transactions on Petri nets and other models of concurrency II*, pages 1–26. Springer, 2009. (Cited on pages 8, 272, and 313.)

[134] Wil M. P. van der Aalst. Data scientist: The engineer of the future. In *Enterprise interoperability VI*, pages 13–26. Springer, 2014. (Cited on page 70.)

[135] Wil M. P. van der Aalst. Extracting event data from databases to unleash process mining. In *BPM-Driving innovation in a digital world*, pages 105–128. Springer, 2015. (Cited on pages 71, 78, and 118.)

[136] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, 2016. (Cited on pages xvii, xxvi, 3, 8, 94, 95, 213, 220, 272, and 313.)

[137] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and

performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. (Cited on pages 224, 225, 233, 268, 272, and 313.)

[138] Wil M. P. van der Aalst, Alessandro Artale, Marco Montali, and Simone Tritini. Object-centric behavioral constraints: Integrating data and declarative process modelling. In *Description Logics*, 2017. (Cited on page 124.)

[139] Wil M. P. van der Aalst, Paulo Barthelmess, Clarence A. Ellis, and Jacques Wainer. Workflow modeling using proclets. In *International Conference on Cooperative Information Systems*, pages 198–209. Springer, 2000. (Cited on pages 41, 42, and 163.)

[140] Wil M. P. van der Aalst, Paulo Barthelmess, Clarence A. Ellis, and Jacques Wainer. Proclets: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, 10(04):443–481, 2001. (Cited on pages 41, 42, 123, and 163.)

[141] Wil M. P. van der Aalst, Guangming Li, and Marco Montali. Object-Centric Behavioral Constraints. Corr technical report, arXiv.org e-Print archive, 2017. Available at https://arxiv.org/abs/1703.05740. (Cited on pages 119, 120, and 124.)

[142] Wil M. P. van der Aalst and Maja Pesic. DecSerFlow: Towards a truly declarative service flow language. In *International Workshop on Web Services and Formal Methods*, pages 1–23. Springer, 2006. (Cited on page 38.)

[143] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science-Research and Development*, 23(2):99–113, 2009. (Cited on pages 124, 133, and 166.)

[144] Wil M. P. van der Aalst and Christian Stahl. *Modeling business processes: a Petri net-oriented approach*. MIT press, 2011. (Cited on pages 33 and 256.)

[145] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: yet another workflow language. *Information systems*, 30(4):245–275, 2005. (Cited on page 314.)

[146] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Mathias Weske. Business process management: A survey. In *Proceedings of International*

Conference on Business Process Management (BPM), page 1–12, 2003. (Cited on page 8.)

[147] Wil M. P. van der Aalst and Boudewijn F. van Dongen. Discovering workflow performance models from timed logs. In *Engineering and Deployment of Cooperative Information Systems*, pages 45–63. Springer, 2002. (Cited on pages 272 and 313.)

[148] Wil M. P. van der Aalst and Kees M. van Hee. Business process redesign: A Petri-net-based approach. *Computers in industry*, 29(1-2):15–26, 1996. (Cited on page 122.)

[149] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow management: models, methods, and systems*. MIT press, 2004. (Cited on pages 8, 216, and 220.)

[150] Wil M. P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005. (Cited on page 162.)

[151] Jan M. E. M. van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. In *International conference on applications and theory of petri nets*, pages 368–387. Springer, 2008. (Cited on page 216.)

[152] Boudewijn F. van Dongen and Wil M. P. van der Aalst. A meta model for process mining data. *EMOI-INTEROP*, 160:30, 2005. (Cited on page 50.)

[153] Maikel L. van Eck, Natalia Sidorova, and Wil M. P. van der Aalst. Discovering and exploring state-based models for multi-perspective processes. In *International Conference on Business Process Management*, pages 142–157. Springer, 2016. (Cited on pages 218 and 220.)

[154] Kees M. van Hee. *Information System Engineering: A Formal Approach*. Cambridge University Press, 1994. (Cited on pages 123, 161, and 162.)

[155] Sebastiaan J. van Zelst, Boudewijn F. van Dongen, Wil M. P. van der Aalst, and H. M. W. (Eric) Verbeek. Discovering workflow nets using integer linear programming. *Computing*, 100(5):529–556, 2018. (Cited on page 220.)

[156] Irene Vanderfeesten, Hajo A. Reijers, and Wil M. P. van der Aalst. Product-based workflow support. *Information Systems*, 36(2):517–535, 2011. (Cited on page 162.)

[157] H. M. W. (Eric) Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, Wil M. P. van der Aalst, et al. XES, XESame, and ProM 6. In *CAiSE Forum*, volume 72, pages 60–75. Springer, 2010. (Cited on pages 50 and 118.)

[158] Peter A. C. Verkoulen. *Integrated Information Systems Design: An Approach Based on Object-Oriented Concepts and Petri Nets*. PhD thesis, Eindhoven University of Technology, Eindhoven, 1993. (Cited on page 162.)

[159] Anton J. M. M. Weijters and Joel T. S. Ribeiro. Flexible heuristics miner (FHM). In *Computational Intelligence and Data Mining (CIDM), 2011 IEEE Symposium on*, pages 310–317. IEEE, 2011. (Cited on pages 216 and 220.)

[160] Anton J. M. M. Weijters and Wil M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003. (Cited on page 216.)

[161] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. (Cited on page 8.)

[162] Cristian R. Zervos. *Coloured Petri Nets: Their Properties and Applications*. PhD thesis, University of Michigan, Michigan, 1977. (Cited on page 33.)

[163] Cristian R. Zervos and Keki B. Irani. Colored Petri nets: Their properties and applications. Technical report, MICHIGAN UNIV ANN ARBOR SYSTEMS ENGINEERING LAB, 1977. (Cited on page 162.)

[164] Meihui Zhang, Marios Hadjieleftheriou, Beng C. Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. On multi-column foreign key discovery. *Proceedings of the VLDB Endowment*, 3(1-2):805–814, 2010. (Cited on page 219.)

[165] Deng Zhao, Walid Gaaloul, Wenbo Zhang, Chunsheng Zhu, and Zhangbing Zhou. Formal verification of temporal constraints for mobile service-based business process models. *IEEE access*, 6:59843–59852, 2018. (Cited on page 268.)

[166] Michael Zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *Seminal Contributions to Information Systems Engineering*, pages 429–443. Springer, 2013. (Cited on page 37.)

Summary

Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models

In today's world, enterprises are using information systems to support the execution of their business processes. The amount of data being stored about process executions is rapidly growing. Process mining appears to leverage execution data to analyze the business processes executed on information systems. Existing process mining techniques make a fundamental assumption about processes: process models and event logs assume the presence of a well-defined case notion. This implies that each event refers to a case and that the model describes the life-cycle of cases. This assumption is consistent with case-centric information systems. However, most information systems one encounters in enterprises nowadays are artifact-centric, which do not assume a case notion in their business processes. Such differences lead to problems when applying existing process mining techniques, e.g., it is difficult to identify the case notion for the whole process, and the many-to-many relations cannot be well described.

In this thesis, we addressed these problems by proposing a series of process mining techniques based on a novel type of models, i.e., Object-Centric Behavioral Constraint (OCBC) Models. An OCBC model combines data/object models (ER, UML, or ORM) and declarative models (Declare), which can describe both data and behavioral perspectives of business processes in one single diagram. Based on the model, various techniques ranging from event log extraction to performance analysis were created, which are summarized as follows.

The data generated by artifact-centric information systems are different from case-centric event data and have their own features. Therefore, Chapter 3 defines the notion of object-centric event data to represent the data from artifact-centric systems. Subsequently, Chapter 4 introduces the XOC log format, which can better cater to data from artifact-centric systems, and gives an approach to

automatically extract XOC logs from object-centric event data.

Chapter 5 illustrates the novel OCBC modeling language by explaining all the involved elements. The data perspective (e.g., classes) is first explained, followed by the behavioral perspective (e.g., activities). Then these two perspectives are combined, resulting in an OCBC model. Chapter 6 presents approaches to automatically discover OCBC models from XOC logs. First, a basic approach is illustrated to discover models from clean logs. Then, a more robust discovery approach is proposed to deal with noise in real life data.

Based on an XOC log and a manually designed or discovered OCBC model, Chapter 7 shows techniques to diagnose the conformance between them. By taking the data perspective into consideration, it is possible to detect and diagnose a range of conformance problems that would have remained undetected by conventional approaches. Besides, metrics such as fitness, precision and generalization are defined to quantify the degree of conformance. Chapter 8 analyzes the performance in terms of frequencies and times by taking an XOC log and an OCBC model as input. In order to show performance results from different angles, (dotted and column) charts, and indicators are used. With the obtained results in hands, it is possible to map them onto the model, and from this step important bottlenecks could be revealed.

Finally, Chapter 9 presents a case study to apply the proposed techniques to real-life data. It shows the methodology to link all techniques in a consistent manner. Subsequently, Chapter 10 summarizes this thesis by discussing contributions and limitations. Besides, it also sketches ideas for further research in OCBC techniques.

Acknowledgments

I am very glad to pursue a PhD at TU/e in the Netherlands. This period of time has changed my life and will be in my memories forever. During the PhD, I have been helped by a lot of people. Now it is time to express my gratitude.

First of all, I would like to thank my first promotor, Wil van der Aalst. You gave me invaluable supervision using your wealth of knowledge and deep intuition. You guided me to overcome many problems in patience and my PhD would not have finished without your help. What I have learned from you is not only research but also the method to do research. Secondly, but not less important, I thank my co-promotor Renata Medeiros de Carvalho. You are always the first one I "bother" when I have questions in research. You spent a lot of time to discuss with me and helped me to structure my thoughts. You are a very kind person and a good friend to me, and I always feel comfortable to talk to you.

I thank my reading committee members Boudewijn van Dongen, George Fletcher, Marco Montali, Hafedh Mili and Walid Gaaloul for giving me helpful comments and attending my defense.

I want to thank Shengnan for picking me up at the train station and helping me settle down on my first day in Eindhoven. You introduced me to the group and the city, such that I could get familiar with the new environment soon. I also thank Xixi, Jianpeng, Yulong, Cong and Xin for sharing the joy and burdens during our PhD adventures. Special thanks to Long and Fei. I enjoyed talking to you during lunch.

I would like to thank people who shared an office with me. Thank you Elham, Maikel Leemans and Dennis. You supported me a lot when I just started in the group. I also thank Vadim, Ayda, Mitchel, Marie, Hilda and Murian for creating a nice atmosphere in the office. Many thanks to Ine, Riet and José. You are always willing to help everyone in our group and make our lives easier. Special thanks to Ine for helping me contact committee members and proofreading my

thesis. I also would like to thank all my other colleagues: Massimiliano, Eric, Dirk, Natalia, Marwan, Farideh, Murat, Joos, Eduardo, Bas, Felix, Alfredo, Alok, Maikel van E., Bart, Niek, Sander, Shiva, Alifah, Nour, Marcus, Rashid, Azadeh, Wouter, Natasha, Shiwei, Tianjin, Lu, Kaijie, Yuhao, Ricky, Anil, Simon, Mykola, Pieter, Tita, Loek, Wilco, Xuming and the rest.

I thank my roommate Qiang, who provided a warm and pleasant living environment for me. Via your introduction, I made a lot of new friends. Many thanks to Liangliang, Haiyu, Fulong, Wang, Shuli, Teng, Xuwei, Lu, Chenhui, Bitao, Xiaotao. I really enjoyed the time playing with you all.

Last but not least, I want to thank some people in China. I thank my master supervisor Baoxin Xiu. With your help, I got the chance to pursue my PhD abroad. I also thank my brother Guanghui. You gave me useful suggestions when I was choosing universities for my PhD. My deepest gratitude goes to my parents for your support throughout my life. You were aware of the importance of education and offered me all that you can towards realizing this.

Guangming
Eindhoven, April 2019

Curriculum Vitae

Guangming Li was born on April 15th, 1989 and grew up in Kaifeng, China. In 2012, he received his bachelor degree in Electrical Engineering from Harbin Institute of Technology, Harbin, China. He has a Master of Engineering in Management Science and Engineering from the National University of Defense Technology in Changsha, China, which he received in 2014. Since September 2014 he has worked as a PhD student in the Department of Mathematics and Computer Science at Eindhoven University of Technology, the Netherlands, from which he achieved the results presented in this dissertation.

Guangming Li has received the following awards:

- The *Best Paper Award* at the 20th International Conference on Business Information Systems in 2017 for the paper *Automatic discovery of object-centric behavioral constraint models*.

List of Publications

Guangming Li has the following publications:

Journals

- **Guangming Li** and Wil M. P. van der Aalst. A framework for detecting deviations in complex event logs. *Intelligent Data Analysis*, 21(4):759-779, 2017.

- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. Discovery of object-centric behavioral constraint models based on customized metrics. *Information Sciences*. Under review. (Corresponding to Chapter 6)

- **Guangming Li** and Renata M. de Carvalho. Process mining in social media: Applying object-centric behavioral constraint models. *IEEE Access*. Under review. (Corresponding to Chapters 8 and 9)

Proceedings and Workshop Contributions

- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. Automatic discovery of object-centric behavioral constraint model. In *International Conference on Business Information Systems*, pages 43-58. Springer, 2017. (Corresponding to Chapter 6)

- **Guangming Li**, Eduardo González López de Murillas, Renata M. de Carvalho, and Wil M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Information Systems in the Big Data Era*, pages 182–199. Springer, 2018. (Corresponding to Chapter 4)

- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. Configurable event correlation for process discovery from object-centric event data. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 203-210. IEEE, 2018. (Corresponding to Chapter 3)
- **Guangming Li** and Renata M. de Carvalho. Dealing with artifact-centric systems: A process mining approach. In *EMISA 2018*, pages 80-84. (Corresponding to Chapter 1)
- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. Object-centric behavioral constraint models: A hybrid model for behavioral and data perspectives. In *The 34th ACM/SIGAPP Symposium On Applied Computing (SAC 2019)*, pages 48-56. ACM, 2019. (Corresponding to Chapter 5)
- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. A model-based framework to automatically generate semi-real data for evaluating data analysis techniques. In *International Conference on Enterprise Information Systems*. SCITEPRESS, 2019. (Corresponding to Chapter 3)

Technical Reports

- Wil M. P. van der Aalst, **Guangming Li**, and Marco Montali. Object-centric behavioral constraints. CoRR technical report. March 2017. (Corresponding to Chapter 7)

Websites, Software, and Data Sets

- **Guangming Li**, Renata M. de Carvalho, and Wil M. P. van der Aalst. A website for OCBC techniques, documents and software. https://www.win.tue.nl/ocbc/.
- **Guangming Li**. OCBC Model Editor. Available at the OCBC website.
- **Guangming Li**. OCBC Model Miner. Available at the OCBC website.
- **Guangming Li**. OCBC Conformance Checker. Available at the OCBC website.
- **Guangming Li**. ERP Data Generator. Available at the OCBC website.
- **Guangming Li**. Stack Exchange Data Set (parsed to CSV files). Available at the OCBC website.

SIKS dissertations

2011

01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*
02 Nick Tinnemeier(UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*
03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*
04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference.*
05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline..*
06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*
07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*
08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*
09 Tim de Jong (OU), *Contextualised Mobile Media for Learning.*
10 Bart Bogaert (UvT), *Cloud Content Contention.*
11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*
12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*
13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*
14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*
15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*
16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*
17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*
18 Mark Ponsen (UM), *Strategic Decision-Making in complex games.*
19 Ellen Rusman (OU), *The Mind ' s Eye on Personal Profiles.*
20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*
21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented Systems.*
22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*
23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*

24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*

25 Syed Waqar ul Qounain Jaffry (VU)), *Analysis and Validation of Models for Trust Dynamics.*

26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*

27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*

28 Rianne Kaptein(UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*

29 Faisal Kamiran (TUE), *Discrimination-aware Classification.*

30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*

31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*

32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*

33 Tom van der Weide (UU), *Arguing to Motivate Decisions.*

34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*

35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training.*

36 Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*

37 Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*

38 Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*

39 Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*

40 Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*

41 Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*

42 Michal Sindlar (UU), *Explaining Behavior through Mental State Attribution.*

43 Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*

44 Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*

45 Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*

46 Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*

47 Azizi Bin Ab Aziz(VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*

48 Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*

49 Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*

2012

01 Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*

02 Muhammad Umair(VU), *Adaptivity, emotion, and Rationality in Human and*

Ambient Agent Models.

03 Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*

04 Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*

05 Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*

06 Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*

07 Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*

08 Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*

09 Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*

10 David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*

11 J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*

12 Kees van der Sluijs (TUE), *Model Driven Design and Data Integration in Semantic Web Information Systems.*

13 Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*

14 Evgeny Knutov(TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*

15 Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*

16 Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*

17 Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*

18 Eltjo Poort (VU), *Improving Solution Architecting Practices.*

19 Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*

20 Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*

21 Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*

22 Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*

23 Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*

24 Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*

25 Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*

26 Emile de Maat (UVA), *Making Sense of Legal Text.*

27 Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*

28 Nancy Pascall (UvT), *Engendering Technology Empowering Women.*

29 Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*

30 Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*

31 Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Ca-*

pacity and Infrastructure.

32 Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*

33 Rory Sie (OUN), *Coalitions in Cooperation Networks (COCOON).*

34 Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and applications.*

35 Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*

36 Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*

37 Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*

38 Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*

39 Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*

40 Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*

41 Sebastian Kelle (OU), *Game Design Patterns for Learning.*

42 Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning.*

43 Withdrawn, .

44 Anna Tordai (VU), *On Combining Alignment Techniques.*

45 Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*

46 Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*

47 Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*

48 Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*

49 Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*

50 Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering.*

51 Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.*

2013

01 Viorel Milea (EUR), *News Analytics for Financial Decision Support.*

02 Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*

03 Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*

04 Chetan Yadati(TUD), *Coordinating autonomous planning and scheduling.*

05 Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*

06 Romulo Goncalves(CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*

07 Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*

08 Robbert-Jan Merk(VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*

09 Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*

10 Jeewanie Jayasinghe Arachchige(UvT), *A Unified Modeling Framework for Service Design..*

11 Evangelos Pournaras(TUD), *Multi-level Reconfigurable Self-organization in Overlay*

Services.

12 Marian Razavian(VU), *Knowledge-driven Migration to Services.*

13 Mohammad Safiri(UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*

14 Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*

15 Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications.*

16 Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation.*

17 Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*

18 Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification.*

19 Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling.*

20 Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval.*

21 Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation.*

22 Tom Claassen (RUN), *Causal Discovery and Logic.*

23 Patricio de Alencar Silva(UvT), *Value Activity Monitoring.*

24 Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning.*

25 Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System.*

26 Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning.*

27 Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance.*

28 Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience.*

29 Iwan de Kok (UT), *Listening Heads.*

30 Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support.*

31 Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications.*

32 Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.*

33 Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere.*

34 Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search.*

35 Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction.*

36 Than Lam Hoang (TUe), *Pattern Mining in Data Streams.*

37 Dirk Börner (OUN), *Ambient Learning Displays.*

38 Eelco den Heijer (VU), *Autonomous Evolutionary Art.*

39 Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems.*

40 Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games.*

41 Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.*

42 Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning.*

43 Marc Bron (UVA), *Exploration and Contextualization through Interaction and Concepts.*

2014

01 Nicola Barile (UU), *Studies in Learning Monotone Models from Data.*
02 Fiona Tuliyano (RUN), *Combining System Dynamics with a Domain Modeling Method.*
03 Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions.*
04 Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.*
05 Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability.*
06 Damian Tamburri (VU), *Supporting Networked Software Development.*
07 Arya Adriansyah (TUE), *Aligning Observed and Modeled Behavior.*
08 Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints.*
09 Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.*
10 Ivan Salvador Razo Zapata (VU), *Service Value Networks.*
11 Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support.*
12 Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control.*
13 Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.*
14 Yangyang Shi (TUD), *Language Models With Meta-information.*
15 Natalya Mogles (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*
16 Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*
17 Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*
18 Mattijs Ghijsen (VU), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*
19 Vincius Ramos (TUE), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*
20 Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*
21 Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments.*
22 Marieke Peeters (UT), *Personalized Educational Games - Developing agent-supported scenario-based training.*
23 Eleftherios Sidirourgos (UvA/CWI), *Space Efficient Indexes for the Big Data Era.*
24 Davide Ceolin (VU), *Trusting Semi-structured Web Data.*
25 Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*
26 Tim Baarslag (TUD), *What to Bid and When to Stop.*
27 Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*
28 Anna Chmielowiec (VU), *Decentralized k-Clique Matching.*
29 Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*
30 Peter de Kock Berenschot (UvT), *Anticipating Criminal Behaviour.*
31 Leo van Moergestel (UU), *Agent Tech-*

nology in Agile Multiparallel Manufacturing and Product Support.

32 Naser Ayat (UVA), *On Entity Resolution in Probabilistic Data.*

33 Tesfa Tegegne Asfaw (RUN), *Service Discovery in eHealth.*

34 Christina Manteli (VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.*

35 Joost van Oijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach.*

36 Joos Buijs (TUE), *Flexible Evolutionary Algorithms for Mining Structured Process Models.*

37 Maral Dadvar (UT), *Experts and Machines United Against Cyberbullying.*

38 Danny Plass-Oude Bos (UT), *Making brain-computer interfaces better: improving usability through post-processing..*

39 Jasmina Maric (UvT), *Web Communities, Immigration, and Social Capital.*

40 Walter Omona (RUN), *A Framework for Knowledge Management Using ICT in Higher Education.*

41 Frederic Hogenboom (EUR), *Automated Detection of Financial Events in News Text.*

42 Carsten Eijckhof (CWI/TUD), *Contextual Multidimensional Relevance Models.*

43 Kevin Vlaanderen (UU), *Supporting Process Improvement using Method Increments.*

44 Paulien Meesters (UvT), *Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden..*

45 Birgit Schmitz (OUN), *Mobile Games for Learning: A Pattern-Based Approach.*

46 Ke Tao (TUD), *Social Web Data Analytics: Relevance, Redundancy, Diversity.*

47 Shangsong Liang (UVA), *Fusion and Diversification in Information Retrieval.*

2015

01 Niels Netten (UvA), *Machine Learning for Relevance of Information in Crisis Response.*

02 Faiza Bukhsh (UvT), *Smart auditing: Innovative Compliance Checking in Customs Controls.*

03 Twan van Laarhoven (RUN), *Machine learning for network data.*

04 Howard Spoelstra (OUN), *Collaborations in Open Learning Environments.*

05 Christoph Bösch(UT), *Cryptographically Enforced Search Pattern Hiding.*

06 Farideh Heidari (TUD), *Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes.*

07 Maria-Hendrike Peetz(UvA), *Time-Aware Online Reputation Analysis.*

08 Jie Jiang (TUD), *Organizational Compliance: An agent-based model for designing and evaluating organizational interactions.*

09 Randy Klaassen(UT), *HCI Perspectives on Behavior Change Support Systems.*

10 Henry Hermans (OUN), *OpenU: design of an integrated system to support lifelong learning.*

11 Yongming Luo(TUE), *Designing algorithms for big graph datasets: A study of computing bisimulation and joins.*

12 Julie M. Birkholz (VU), *Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks.*

13 Giuseppe Procaccianti(VU), *Energy-Efficient Software.*

14 Bart van Straalen (UT), *A cognitive approach to modeling bad news conversations.*

15 Klaas Andries de Graaf (VU), *Ontology-based Software Architecture Documentation.*

16 Changyun Wei (UT), *Cognitive Coordination for Cooperative Multi-Robot Teamwork.*

17 André van Cleeff (UT), *Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs.*
18 Holger Pirk (CWI), *Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories.*
19 Bernardo Tabuenca (OUN), *Ubiquitous Technology for Lifelong Learners.*
20 Loïs Vanhée(UU), *Using Culture and Values to Support Flexible Coordination.*
21 Sibren Fetter (OUN), *Using Peer-Support to Expand and Stabilize Online Learning.*
22 Zhemin Zhu(UT), *Co-occurrence Rate Networks.*
23 Luit Gazendam (VU), *Cataloguer Support in Cultural Heritage.*
24 Richard Berendsen (UVA), *Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation.*
25 Steven Woudenberg (UU), *Bayesian Tools for Early Disease Detection.*
26 Alexander Hogenboom (EUR), *Sentiment Analysis of Text Guided by Semantics and Structure.*
27 Sándor Héman (CWI), *Updating compressed colomn stores.*
28 Janet Bagorogoza(TiU), *KNOWLEDGE MANAGEMENT AND HIGH PERFORMANCE; The Uganda Financial Institutions Model for HPO.*
29 Hendrik Baier (UM), *Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains.*
30 Kiavash Bahreini(OU), *Real-time Multimodal Emotion Recognition in E-Learning.*
31 Yakup Koç (TUD), *On the robustness of Power Grids.*
32 Jerome Gard(UL), *Corporate Venture Management in SMEs.*
33 Frederik Schadd (TUD), *Ontology Mapping with Auxiliary Resources.*
34 Victor de Graaf(UT), *Gesocial Recommender Systems.*
35 Jungxao Xu (TUD), *Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction.*

2016

01 Syed Saiden Abbas (RUN), *Recognition of Shapes by Humans and Machines.*
02 Michiel Christiaan Meulendijk (UU), *Optimizing medication reviews through decision support: prescribing a better pill to swallow.*
03 Maya Sappelli (RUN), *Knowledge Work in Context: User Centered Knowledge Worker Support.*
04 Laurens Rietveld (VU), *Publishing and Consuming Linked Data.*
05 Evgeny Sherkhonov (UVA), *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers.*
06 Michel Wilson (TUD), *Robust scheduling in an uncertain environment.*
07 Jeroen de Man (VU), *Measuring and modeling negative emotions for virtual training.*
08 Matje van de Camp (TiU), *A Link to the Past: Constructing Historical Social Networks from Unstructured Data.*
09 Archana Nottamkandath (VU), *Trusting Crowdsourced Information on Cultural Artefacts.*
10 George Karafotias (VUA), *Parameter Control for Evolutionary Algorithms.*
11 Anne Schuth (UVA), *Search Engines that Learn from Their Users.*
12 Max Knobbout (UU), *Logics for Modelling and Verifying Normative Multi-Agent Systems.*
13 Nana Baah Gyan (VU), *The Web, Speech*

Technologies and Rural Development in West Africa - An ICT4D Approach.

14 Ravi Khadka (UU), *Revisiting Legacy Software System Modernization.*

15 Steffen Michels (RUN), *Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments.*

16 Guangliang Li (UVA), *Socially Intelligent Autonomous Agents that Learn from Human Reward.*

17 Berend Weel (VU), *Towards Embodied Evolution of Robot Organisms.*

18 Albert Meroño Peñuela (VU), *Refining Statistical Data on the Web.*

19 Julia Efremova (Tu/e), *Mining Social Structures from Genealogical Data.*

20 Daan Odijk (UVA), *Context & Semantics in News & Web Search.*

21 Alejandro Moreno Celleri (UT), *From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground.*

22 Grace Lewis (VU), *Software Architecture Strategies for Cyber-Foraging Systems.*

23 Fei Cai (UVA), *Query Auto Completion in Information Retrieval.*

24 Brend Wanders (UT), *Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach.*

25 Julia Kiseleva (TU/e), *Using Contextual Information to Understand Searching and Browsing Behavior.*

26 Dilhan Thilakarathne (VU), *In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains.*

27 Wen Li (TUD), *Understanding Geospatial Information on Social Media.*

28 Mingxin Zhang (TUD), *Large-scale Agent-based Social Simulation - A study on epidemic prediction and control.*

29 Nicolas Höning (TUD), *Peak reduction in decentralised electricity systems -Markets and prices for flexible planning.*

30 Ruud Mattheij (UvT), *The Eyes Have It.*

31 Mohammad Khelghati (UT), *Deep web content monitoring.*

32 Eelco Vriezekolk (UT), *Assessing Telecommunication Service Availability Risks for Crisis Organisations.*

33 Peter Bloem (UVA), *Single Sample Statistics, exercises in learning from just one example.*

34 Dennis Schunselaar (TUE), *Configurable Process Trees: Elicitation, Analysis, and Enactment.*

35 Zhaochun Ren (UVA), *Monitoring Social Media: Summarization, Classification and Recommendation.*

36 Daphne Karreman (UT), *Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies.*

37 Giovanni Sileno (UvA), *Aligning Law and Action - a conceptual and computational inquiry.*

38 Andrea Minuto (UT), *MATERIALS THAT MATTER - Smart Materials meet Art & Interaction Design.*

39 Merijn Bruijnes (UT), *Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect.*

40 Christian Detweiler (TUD), *Accounting for Values in Design.*

41 Thomas King (TUD), *Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance.*

42 Spyros Martzoukos (UVA), *Combinatorial and Compositional Aspects of Bilingual Aligned Corpora.*

43 Saskia Koldijk (RUN), *Context-Aware Support for Stress Self-Management: From*

Theory to Practice.

44 Thibault Sellam (UVA), *Automatic Assistants for Database Exploration.*

45 Bram van de Laar (UT), *Experiencing Brain-Computer Interface Control.*

46 Jorge Gallego Perez (UT), *Robots to Make you Happy.*

47 Christina Weber (UL), *Real-time foresight - Preparedness for dynamic innovation networks.*

48 Tanja Buttler (TUD), *Collecting Lessons Learned.*

49 Gleb Polevoy (TUD), *Participation and Interaction in Projects. A Game-Theoretic Analysis.*

50 Yan Wang (UVT), *The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains.*

2017

01 Jan-Jaap Oerlemans (UL), *Investigating Cybercrime.*

02 Sjoerd Timmer (UU), *Designing and Understanding Forensic Bayesian Networks using Argumentation.*

03 Daniël Harold Telgen (UU), *Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines.*

04 Mrunal Gawade (CWI), *MULTI-CORE PARALLELISM IN A COLUMN-STORE.*

05 Mahdieh Shadi (UVA), *Collaboration Behavior.*

06 Damir Vandic (EUR), *Intelligent Information Systems for Web Product Search.*

07 Roel Bertens (UU), *Insight in Information: from Abstract to Anomaly.*

08 Rob Konijn (VU), *Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery.*

09 Dong Nguyen (UT), *Text as Social and Cultural Data: A Computational Perspective on Variation in Text.*

10 Robby van Delden (UT), *(Steering) Interactive Play Behavior.*

11 Florian Kunneman (RUN), *Modelling patterns of time and emotion in Twitter #anticipointment.*

12 Sander Leemans (TUE), *Robust Process Mining with Guarantees.*

13 Gijs Huisman (UT), *Social Touch Technology - Extending the reach of social touch through haptic technology.*

14 Shoshannah Tekofsky (UvT), *You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior.*

15 Peter Berck, Radboud University (RUN), *Memory-Based Text Correction.*

16 Aleksandr Chuklin (UVA), *Understanding and Modeling Users of Modern Search Engines.*

17 Daniel Dimov (UL), *Crowdsourced Online Dispute Resolution.*

18 Ridho Reinanda (UVA), *Entity Associations for Search.*

19 Jeroen Vuurens (TUD), *Proximity of Terms, Texts and Semantic Vectors in Information Retrieval.*

20 Mohammadbashir Sedighi (TUD), *Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility.*

21 Jeroen Linssen (UT), *Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds).*

22 Sara Magliacane (VU), *Logics for causal inference under uncertainty.*

23 David Graus (UVA), *Entities of Interest—Discovery in Digital Traces.*

24 Chang Wang (TUD), *Use of Affordances for Efficient Robot Learning.*
25 Veruska Zamborlini (VU), *Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search.*
26 Merel Jung (UT), *Socially intelligent robots that understand and respond to human touch.*
27 Michiel Joosse (UT), *Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors.*
28 John Klein (VU), *Architecture Practices for Complex Contexts.*
29 Adel Alhuraibi (UVT), *From IT-Business Strategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT.*
30 Wilma Latuny (UVT), *The Power of Facial Expressions.*
31 Ben Ruijl (UL), *Advances in computational methods for QFT calculations.*
32 Thaer Samar (RUN), *Access to and Retrievability of Content in Web Archives.*
33 Brigit van Loggem (OU), *Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity.*
34 Maren Scheffel (OUN), *The Evaluation Framework for Learning Analytics.*
35 Martine de Vos (VU), *Interpreting natural science spreadsheets.*
36 Yuanhao Guo (UL), *Shape Analysis for Phenotype Characterisation from High-throughput Imaging.*
37 Alejandro Montes García (TUE), *WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy.*
38 Alex Kayal (TUD), *Normative Social Applications.*
39 Sara Ahmadi (RUN), *Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR.*
40 Altaf Hussain Abro (VUA), *Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems.*
41 Adnan Manzoor (VUA), *Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle.*
42 Elena Sokolova (RUN), *Causal discovery from mixed and missing data with applications on ADHD datasets.*
43 Maaike de Boer (RUN), *Semantic Mapping in Video Retrieval.*
44 Garm Lucassen (UU), *Understanding User Stories - Computational Linguistics in Agile Requirements Engineering.*
45 Bas Testerink (UU), *Decentralized Runtime Norm Enforcement.*
46 Jan Schneider (OU), *Sensor-based Learning Support.*
47 Yie Yang (TUD), *Crowd Knowledge Creation Acceleration.*
48 Angel Suarez (OU), *Colloborative inquiry-based learning.*

2018

01 Han van der Aa (VUA), *Comparing and Aligning Process Representations.*
02 Felix Mannhardt (TUE), *Multi-perspective Process Mining.*
03 Steven Bosems (UT), *Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction.*
04 Jordan Janeiro (TUD), *Flexible Coordination Support for Diagnosis Teams in*

Data-Centric Engineering Tasks.

05 Hugo Huurdeman (UVA), *Supporting the Complex Dynamics of the Information Seeking Process.*

06 Dan Ionita (UT), *Model-Driven Information Security Risk Assessment of Socio-Technical Systems.*

07 Jieting Luo (UU), *A formal account of opportunism in multi-agent systems .*

08 Rick Smetsers (RUN), *Advances in Model Learning for Software Systems.*

09 Xu Xie (TUD), *Data Assimilation in Discrete Event Simulations.*

10 Julia S. Mollee (VU), *Moving forward: supporting physical activity behavior change through intelligent technology.*

11 Mahdi Sargolzaei (UVA), *Enabling Framework for Service-oriented Collaborative Networks.*

12 Xixi Lu (TUE), *Using Behavioral Context in Process Mining.*

13 Seyed Amin Tabatabaei (VUA), *Computing a Sustainable Future: Exploring the added value of computational models for increasing the use of renewable energy in the residential sector.*

14 Bart Joosten (UVT), *Detecting Social Signals with Spatiotemporal Gabor Filters.*

15 Naser Davarzani (UM), *Biomarker discovery in heart failure.*

16 Jaebok Kim (UT), *Automatic recognition of engagement and emotion in a group of children.*

17 Jianpeng Zhang (TU/e), *On Graph Sample Clustering.*

18 Henriette Nakad (UL), *De Notaris en Private Rechtspraak.*

19 Minh Duc Pham (VUA), *Emergent relational schemas for RDF.*

20 Manxia Liu (RUN), *Time and Bayesian Networks.*

21 Aad Slootmaker (OUN), *EMERGO: a generic platform for authoring and playing scenario-based serious games.*

22 Eric Fernandes de Mello Araujo (VUA), *Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks.*

23 Kim Schouten (EUR), *Semantics-driven Aspect-Based Sentiment Analysis.*

24 Jered Vroon (UT), *Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots.*

25 Riste Gligorov (VUA), *Serious Games in Audio-Visual Collections.*

26 Roelof Anne Jelle de Vries (UT), *Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology.*

27 Maikel Leemans (TUE), *Hierarchical Process Mining for Scalable Software Analysis.*

28 Christian Willemse (UT), *Social Touch Technologies: How they feel and how they make you feel.*

29 Yu Gu (UVT), *Emotion Recognition from Mandarin Speech.*

30 Wouter Beek (VU), *The "K" in "semantic web" stands for "knowledge": scaling semantics to the web.*

2019

01 Rob van Eijk (UL), *Comparing and Aligning Process Representations.*

02 Emmanuelle Beauxis Aussalet (CWI, UU), *Statistics and Visualizations for Assessing Class Size Uncertainty.*

03 Eduardo Gonzalez Lopez de Murillas (TUE), *Process Mining on Databases: Extracting Event Data from Real Life Data Sources.*

04 Ridho Rahmadi (RUN), *Finding stable causal structures from clinical data.*

05 Sebastiaan van Zelst (TUE), *Process Mining with Streaming Data*.

06 Chris Dijkshoorn (VU), *Nichesourcing for Improving Access to Linked Cultural Heritage Datasets*.

07 Soude Fazeli (TUD), *Recommender Systems in Social Learning Platforms*.

08 Frits de Nijs (TUD), *Resource-constrained Multi-agent Markov Decision Processes*.

09 Fahimeh Alizadeh Moghaddam (UVA), *Self-adaptation for energy efficiency in software systems*.

10 Qing Chuan Ye (EUR), *Multi-objective Optimization Methods for Allocation and Prediction*.

11 Yue Zhao (TUD), *Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs*.

12 Jacqueline Heinerman (VU), *Better Together*.

13 Guanliang Chen (TUD), *MOOC Analytics: Learner Modeling and Content Generation*.

14 Daniel Davis (TUD), *Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses*.

15 Erwin Walraven (TUD), *Planning under Uncertainty in Constrained and Partially*.

16 Guangming Li (TUE), ***Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models***.