Combining Blockchain and IoT for business processes deployment and mining

Leyla Moctar M'Baba

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAS010

Thèse de doctorat

Discovering artifact-centric processes from Blockchain applications

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 30 Mai 2024, par

Leyla Moctar M'Baba

Composition du Jury :

Khalil Drira Professeur, Université de Toulouse	Président
Daniela Grigori Professeure, Paris Dauphine	Rapporteur
Djamal Benslimane Professeur, Université de Lyon	Rapporteur
Kais Klai Professeur, Université Sorbonne Paris Nord	Examinateur
Joaquin Garcia-Alfaro Professeur, Télécom SudParis	Examinateur
Walid Gaaloul Professeur, Télécom SudParis	Directeur de thèse
Mohamedade Farouk NANNE Professeur, Université de Nouakchott	Directeur de thèse
Mohamed Sellami Maître de conférences, Télécom SudParis	Invité
Ahmedou Haouba Professeur, Université de Nouakchott	Invité

To my home and the soul of my soul, my mother,
to my heaven, my grandmother,
to my hero, my father,
to my world, my siblings,
for their prayers, unconditional love and support.

Acknowledgment

I would like to express my deepest gratitude to the people and organizations without whom this journey would not have been possible.

First and foremost, I extend my heartfelt thanks to my supervisors, Prof. Walid Gaalou and Prof. Mohamedade Farouk NANNE, for their unwavering support, insightful guidance, and invaluable mentorship throughout this research journey. Their expertise and encouragement have been pivotal in shaping my academic and personal growth. I am also grateful to my advisor, Dr. Mohamed Sellami, for his patience, encouragement and expertise as well as his wisdom.

I am profoundly grateful to my dissertation committee members, Prof. Khalil Drira, Prof. Daniela Grigori, Prof. Djamal Benslimane, Prof. Kais Klai, Prof. Ahmedou Haouba and Prof. Joaquin Garcia-Alfaro, for their constructive feedback, thoughtful advice, and immense patience. Their perspectives and critiques have immensely contributed to the depth and rigor of my research.

My appreciation also goes out to the staff of the Télécom SudParis, Institut Polytechnique de Paris and University of Nouakchott Al Aasriya, whose dedication to fostering a stimulating and nurturing academic environment has been instrumental in my development as a researcher.

I am indebted to my fellow Ph.D. candidates and research group members for their camaraderie, stimulating discussions, and for being a source of inspiration and motivation. Their friendship and support have been a cornerstone of my Ph.D. experience. In particular, i am indebted to Dr. Nour Assy for her precious advice and help during our collaborations.

Special thanks are due to SCAC, for their financial support through mobility.

I am also grateful to the many authors and researchers whose works have informed and inspired my studies. Their contributions to the field have been invaluable to my research journey.

On a personal note, I would like to express my deepest appreciation to my family: my grandmother, Oumou ElMoueminin, my parents, Mohamed Lemine Moctar M'Baba and Khadijettou Cheibete, and my siblings, Esmaou, Melika and Tijani, for

their unconditional love, immense patience, unwavering belief in me, and constant encouragement. Their sacrifices and support have been the foundation upon which I have built my aspirations and achievements.

Lastly, I would like to extend my thanks to my friends, especially Alia and Smj, both within and outside the academic community, for their understanding, support, and encouragement throughout this challenging yet rewarding journey.

This accomplishment is not just my own but also belongs to all those who have supported me along the way. Thank you all.

Résumé étendu

La blockchain, la technologie derrière les crypto-monnaies, a été initialement utilisée pour effectuer des transactions financières dans des environnements à faible confiance sans autorité centrale. Peu de temps après, la deuxième génération de plateformes blockchain a étendu son applicabilité au-delà des cas d'utilisation financière. La communauté de la gestion des processus métiers (BPM), en particulier, a vu des opportunités pour améliorer le cadre BPM existant et renforcer les collaborations inter-organisationnelles en utilisant la blockchain.

Bien que l'exécution des processus métiers sur la blockchain ait été largement explorée, la recherche sur le process mining des données d'exécution résultantes commence tout juste à gagner du terrain. Les défis liés à la discordance entre les structures de données de la blockchain et les logs de données d'événements (event data) ont été abordés dans la littérature. Cependant, tous les travaux existants se concentraient sur les processus centrés sur les activités et ignoraient les processus centrés sur les artifacts, qui sont fréquents parmi les applications blockchain. Un autre défi est l'extraction des données d'événements centrées sur les artifacts des applications blockchain. Les structures de données et les systèmes de logging de la blockchain ne sont pas intrinsèquement orientés processus. Les données et les logs générés par les applications blockchain sont souvent fragmentés et non structurés, nécessitant une nouvelle approche pour mapper les données de la blockchain aux éléments de données d'événements centrés sur les arteficts. Cette approche doit tenir compte des caractéristiques uniques des données de la blockchain, y compris leur nature cryptographique et l'absence d'identificateurs explicites d'instances de processus.

L'intégration du process mining centré sur les artifacts avec la technologie blockchain présente des défis, tel que le besoin d'un format de stockage approprié. Les formats existants, comme XES, ne sont pas conçus pour capturer les complexités des processus centrés sur les artifacts. Ils entraînent souvent de la redondance et de l'inefficacité. L'utilisation de ces formats de logging centré sur les activités pour stocker des données d'événements centrées sur les artifacts entraîne des problèmes de convergence, de divergence, de dénormalisation et de perte d'informations relatives

à l'évolution des artifacts, à leurs relations et à leurs interactions. Ces problèmes sont à l'origine de résultats de process mining erronés et non conformes à la réalité. Ils ont été partiellement résolus par l'introduction d'OCEL, un standard de logging pour stocker des données d'événements centrées sur les objets. OCEL permet le stockage de données d'objets et leur lien avec des événements, mais ne prend pas en charge l'évolution des objets ni leurs relations. Il est donc nécessaire d'introduire un nouveau format de logging pour capturer efficacement les données d'événements centrées sur les artifacts, en soutenant l'évolution des objets et des relations sans redondance.

La découverte des modèles de processus centrés sur les artifacts à partir du nouveau format de logging proposé est également un défi crucial. Les techniques de découverte existantes ne sont pas compatibles avec les logs centrés sur les artifacts et nécessitent souvent un prétraitement et des connaissances de domaine approfondies. Ainsi, Une nouvelle technique de découverte est nécessaire pour automatiser l'extraction des modèles de processus centrés sur les artifacts directement à partir du format de journal proposé. Cette technique doit être capable d'identifier les conditions de données, les structures hiérarchiques des activités et les interactions entre les artifacts.
Pour relever tous ces défis et débloquer le process mining centré sur les artifacts pour les applications blockchain, nous avons mené les travaux présentés dans ce manuscrit. Nous avons d'abord examiné, à travers une revue systématique, tous les défis concernant la collecte de données d'événements pour le process mining à partir des applications blockchain. Ensuite, nous proposons ACEL, une extension d'OCEL qui résout tous les problèmes liés au stockage des données d'événements centrées sur les artifacts. Ce format soutient l'évolution des objets et de leurs relations, introduisant le concept d'évolution des relations pour améliorer l'identification des dépendances comportementales. Le format est optimisé pour éviter la redondance, garantissant un stockage efficace des données d'événements. Nous proposons également une approche centrée sur les artifacts pour collecter des données d'événements à partir des applications blockchain et les mapper sur des logs ACEL. Cette approche exploite les connaissances de domaine pour interroger les logs de la blockchain et mapper les éléments de données brutes vers les éléments de données d'événements centrés sur les artifacts. La faisabilité de l'approche est évaluée en utilisant deux applications publiques d'Ethereum : Cryptokitties et Augur. Les opportunités de process mining d'ACEL sont d'abord explorées en comparaison avec OCEL en utilisant des techniques existantes de découverte conçues pour OCEL. Puis nous proposons une

nouvelle technique de découverte automatisée pour les modèles de processus centrés sur les artifacts. Cette technique ne nécessite pas de connaissances de domaine et peut fonctionner directement sur le format de journal proposé. Elle combine plusieurs solutions automatisées pour découvrir les conditions de données, les structures hiérarchiques des activités et les interactions entre les artifacts. La validité et la précision de l'approche sont évaluées en utilisant l'application Cryptokitties.

Abstract

Blockchain, the technology behind cryptocurrencies, was initially used for conducting financial transactions in low-trust environments without a central authority. Soon after, the second generation of blockchain platforms expanded its applicability beyond financial use cases. The BPM community, in particular, foresaw opportunities for improving the existing BPM framework and enhancing inter-organizational collaborations using blockchain.

While the execution of business processes on blockchain has been greatly explored, research on process mining of the resulting execution data has just started gaining momentum. The challenges of the mismatch of blockchain data structures with event data logs were mostly addressed. However, all the existing works were focused on activity-centric processes and discarded artifact-centric process, which are frequent among blockchain applications. The activity-centric logging is still used, even outside blockchain, to store artifact-centric event data. Amongst the most used logging formats for this purpose is XES. Using XES to store artifact-centric event data results in convergence, divergence, denormalization and loss of information relating to artifacts' evolution, their relations and interactions. These issues were partly addressed by the introduction of OCEL, a standard to store object-centric event data. OCEL allows the storage of object data and their link to events but does not support object evolution nor object relations.

To address all these challenges and unlock artifact-centric process mining for blockchain applications, we conducted the work presented in the following manuscript. We first investigate, through a systematic review, all the challenges regarding the collection of event data for process mining from blockchain applications. Then, we propose ACEL, an extension of OCEL which solves all the issues related to the storage of artifact-centric event data. We also propose an artifact-centric approach to collect event data from blockchain applications and map them to ACEL logs. The feasibility of the approach is evaluated using two public Ethereum applications: Cryptokitties and Augur. Process mining opportunities of ACEL are first explored in comparison to OCEL by using OCEL-tailored discovery techniques.

Finally, we propose a discovery approach which discovers GSM models, one of the most used modelling languages for artifact-centric processes, directly from ACEL logs. Our approach aims to discover artifact-centric models. To do so, we use a hierarchical clustering of activities based information gain of common conditions. The evaluation of this approach on Cryptokitties demonstrates its feasibility and the benefits in term of accuracy and insights of ACEL for artifact-centric process mining.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Research Context

Processes are everywhere. The presence of any object or entity indicates that there is a creation process that led to its existence. Nature is abundant with processes, from the process followed by plants to convert sunlight, carbon dioxide, and water into glucose and oxygen, to the process followed by bees to produce honey from collected pollen. Humans have learned to replicate the processes they observed in nature, such is the case for agriculture which is, perhaps, one of the earliest cases of process identification and execution. In these early times, the producers were

often the direct consumers, i.e., individuals ran their own production processes. As human groups evolved into organized social structures, specialization emerged, with each group focused on its specific service or product. The second industrial revolution, during the 19th and 20th centuries, solidified this specialization, leading to the creation of managerial positions and functional units in organizations. However, by the late 1980s, the industry realized that their emphasis on functional optimization caused inefficiencies and led to a lost of competitiveness. This realization triggered the development, both in industry and academia, new techniques to improve work flows and processes [1]. Research on the topic demonstrated through empirical studies that process-oriented organizations outperformed non-process-oriented ones. Additionally, the development of new types of IT systems, such as Enterprise Resource Planning (ERP) systems and Workflow Management Systems (WfMSs), provided tools for process improvement and automation. These research and technological advancements led to the emergence of a new discipline known as **Business Process Management (BPM)**. BPM is concerned with understanding and improving business processes to allow organizations to enhance productivity, reduce costs, minimize errors, and deliver higher quality products or services. In the context of BPM, a business process is defined as sequences of activities or tasks that are performed to achieve a specific business goal or objective [2].

Another focus of BPM is the use of technological advancements to improve its set of techniques and toolbox. These advancements came from academia and industry, where solutions, such as Process Aware Information Systems (PAIS) and Business Process Management Systems (BPMS), were developed. Organizations were also on the watch for emerging technologies and trends and ways to use them to their advantage [3]. For example, this was the case with blockchain which was seen as a promise to end the need for a central trust authority to execute transactions. **Blockchain** is a decentralized and distributed ledger technology that enables secure and transparent record-keeping of transactions across a network of computers [4]. The BPM community saw the potential benefits of blockchain for the discipline. Indeed, blockchain can solve the problem of trustworthiness of data and traceability, as it can not be tampered with once it is stored on the blockchain. Blockchain can also contribute to better monitoring as all participants involved in the execution of a process have access to the same state at any given moment. Another benefit of blockchain for BPM is that it allows for the execution of processes on the blockchain using smart contracts [1] as a blockchain-based BPMS. This secure

[1] A smart contract is a program who's code is stored and executed on the blockchain.

execution of processes on blockchain can unlock new types of collaborations where inter-organizational processes can be executed.

BPM encompasses all the aspects of managing a business process through a series of phases called BPM lifecycle [2]. The BPM lifecycle, as illustrated in Figure. 1.1, encompasses the identification, discovery, analysis, redesign, implementation and monitoring phases. The aim of the identification phase is to identify, delimit and inter-relate the processes relevant to solving a business problem. The result of this phase is a new process architecture which provides a global view of all the processes of an organization and their interconnections. The discovery phase provides more details on the previously identified processes by representing them in one or multiple as-is process models. The analysis phase identifies the problems related to previous models and ranks them according to criticality. In the redesign phase, changes to the as-is process models are proposed to solve the previously identified problems to obtain the to-be process. During the implementation phase, the previously selected changes are planned and executed through automation and organizational change management. Finally, the monitoring phase follows the execution of the redesigned process and consists in evaluating its performance at the run-time.

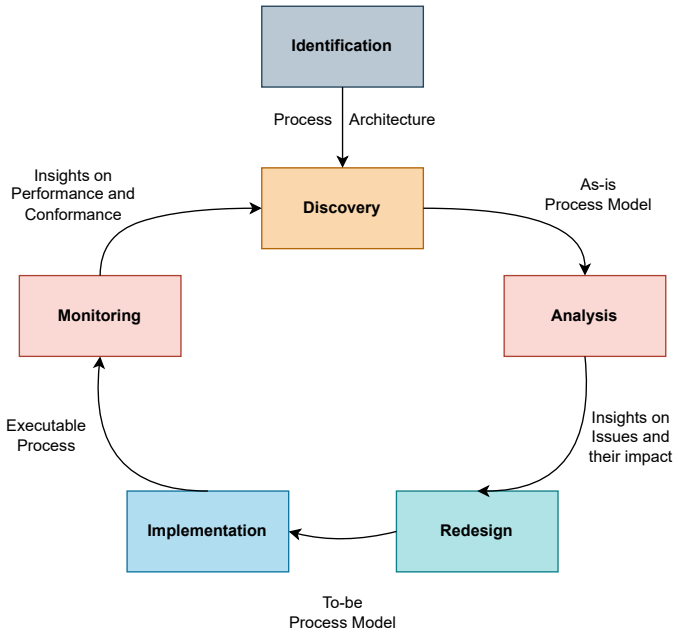

Figure 1.1: Business Process Management Life-cycle

The automation of processes and their digitization continued to increase with the advancement of technology and more organizations relied on process-aware information systems. This digital transformation has led to the generation of big amounts

of event data resulting from the execution of processes. This growing availability of data from various information systems has driven the development of a new discipline within BPM, at the intersection of process science and data science, known as process mining. **Process mining** [5] is a group of techniques that uses event logs from information systems to analyze business processes. It provides insights into how processes are actually executed in practice, offering organizations valuable information to optimize their workflows, identify bottlenecks, and enhance efficiency. Process mining techniques can be grouped, according to their use cases, into four categories: automated process discovery, conformance checking, performance mining, and variants analysis. Automated process discovery techniques fit primarily in the discovery phase of the BPM lifecycle. They use data from event logs to visualize process models depicting the actual execution of processes. In the monitoring phase, performance mining techniques offer real-time tracking of process performance. They can analyze time, cost, and quality parameters to identify deviations from expected performance levels. By identifying process bottlenecks, inefficiencies, and compliance issues, conformance checking techniques directly contribute to the redesign phase of the BPM lifecycle. They provide data-driven insights that guide the modification and optimization of process models to increase performance. Variants analysis techniques also contribute to this phase by identifying, analyzing, and understanding the differences and similarities between various process instances or variants. This analysis helps in uncovering best practices, pinpointing inefficiencies, and identifying opportunities for process optimization or standardization.

Since the emergence of BPM, process modeling consisted mainly in representing tasks/activities and their control flows, thus omitting the data flow of the process execution. This modeling approach, known as the activity-centric approach, where the emphasis is put on how the process operates without details about the execution data, is not very intuitive for business managers [6]. Furthermore, the process models are usually flat, procedural, and imperative which affects the flexibility of the modeling [7]. This gap has been progressively filled over the past years with the introduction of **a new artifact-centric approach**, which takes into account data and process aspects more intuitively for business managers. The modeling languages in this approach use artifacts, which are evolving business entities, as primary modeling concepts and use their lifecycles, which are the business operations that modify the state of artifacts, as a representation of the process model. Artifact-centric modeling languages also support the representation of the interactions between object changes as behavioral inter-dependencies.

The same approach which consists of bringing forward the control flow per-

spective, over the data perspective, has also been dominating the process mining discipline. Additionally, in this activity-centric approach, process mining techniques assume that a process model describes operations related to one independent artifact, which is not true, generally. This realization has sparked a new research area in process mining called *artifact-centric process mining*, where the focus is on the data flow, which aims to solve these issues. In particular, **artifact-centric process discovery** [8] takes as its input either an event log that contains information about events, artifacts and their changes, or a relational database that includes records of data creation, modification, and deletion. The resulting output consists of two key components: a data model and an artifact-centric process model. The data model details the information of each artifact, its attributes and its relationships with other artifacts. The process model delineates the individual lifecycles of artifacts and the interactions between them.

The input of all process mining techniques is event logs. Up until now, the vast majority of works in the literature relied on event logs generated from Process Aware Information Systems (PAISs) [9], Business Process Management Systems (BPMSs) [10] or obtained from relational databases [11]. However, the use of non-conventional data sources, such as blockchain, could be beneficial. Indeed, blockchain constitutes a trustworthy source of immutable execution data and allows the storage of event logs on-chain to ensure their integrity. This helps establish more trust in the process mining results. Furthermore, most existing process mining techniques take an activity-centric approach instead of an artifact-centric one, which is more adapted to real life processes. The novelty of this thesis is that it combines an artifact-centric approach to process mining with the use of blockchain as a data source. This combination comes with many challenges that we will present and illustrate using a motivating example in the following sections.

1.2 Motivating Example

We selected a public blockchain application (Cryptokitties) to serve as an illustrative and motivating example for our approach. This example will also be used to present our approach in the forthcoming chapters.

Cryptokitties [2], is a decentralized application (dApp) built on the Ethereum blockchain, for the breeding and selling of kitties. The journey of a kitty commences with its creation, either as an original entity introduced by the developers at the launch of the application or through special events. These initial creation

[2] `https://www.cryptokitties.co/`

moment marks the entry of each kitty into the digital domain, ready to be adopted by users of the application. Upon acquisition, these virtual kitties become part of a user's collection, each distinguished by unique attributes that set them apart.

Another central phase in the lifecycle of each kitty is the procreation phase where two kitties are paired to produce an offspring. This phase encompasses two activities: a Breeding activity and a Birth activity. The Breeding activity consists of pairing two kitties, owned by a participant or acquired through a transaction with another participant. This pairing is not merely a random combination but a strategic decision influenced by the desire to propagate specific traits or attributes. The underlying mechanism employs a form of digital genetics algorithm which determines the probability of trait inheritance. The act of breeding is subject to a cooldown period, a designed interval that simulates a recovery phase for the involved kitties before they are allowed to breed again. This cooldown phase varies in duration, influenced by the number of Breeding activities a kitty has previously undertaken. Following the Breeding activity, the Birth activity, as the culmination of the genetic combination process, results in the creation of a new kitty. The newborn kitty inherits traits from its parents, with the specific combination of these traits influenced by the underlying digital genetics algorithm. This algorithmic determination results in each newborn kitty possessing a unique set of traits, contributing to the overall diversity within the Cryptokitties ecosystem. The birth of a new kitty not only adds to the participant's collection but also introduces a new entity into the marketplace, potentially possessing rare or desirable traits.

Another integral part of a kitty's lifecycle is the auction phase which involves another business entity called Auction and consists of the trading and exchange of kitties within the kitty marketplace. This phase starts with the Auction Creation activity and ends in either the Auction Completion or the Auction Cancellation activity. The Auction Creation activity involves the creation of an auction entity and setting its specific attributes, such as the starting price, minimum increment, and duration of the auction. These parameters are strategically chosen by the owner to attract potential buyers, balancing the desire for a favorable sale price against the need to ensure the auctioned kitty is appealing to a broad audience. The Auction Completion activity represents the culmination of the sales process, where a transfer of ownership occurs based on the highest bid received. This activity alters the ownership status of the kitty as the ownership is transferred from the seller to the buyer and set the state of the auction entity as successful ending its lifecycle. The Auction Cancellation

activity allows sellers to withdraw their kitty from sale if the auction does not meet their expectations or if they decide against selling. This activity does not alter the ownership status of the kitty but set the state of the auction entity as cancelled representing another possible ending of its lifecycle.

These phases are represented using the activity-centric modeling language BPMN [12] in Figure 1.2. The figure shows the representation of the process from the kitty perspective, without any information regards the attributes and the artifact type. We could only represent the lifecycle of the kitty artifact and not the interaction between the lifecycles of both kitty and auction artifacts. This is due to BPMN using an activity-centric approach for process modelling, which does not show the data perspective contrarily to the artifact-centric approach. Hence, an artifact-centric approach is more compatible with this case. Similarly to Cryptokitties, most blockchain applications revolve around the manipulation of business entities (assets) and depicting their process requires the use of an artifact-centric approach.

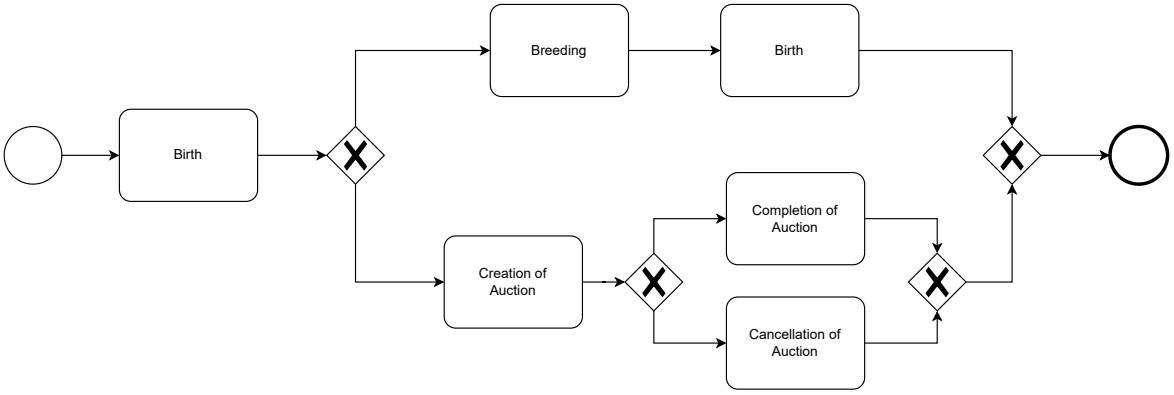

Figure 1.2: Lifecycle of a kitty (BPMN notation)

An excerpt of event logs that could be generated after the execution of the Cryptokitties process is given in Table 1.1, represented using the XES logging [13]. This illustrated event log presents multiple events from the same activities occurring at the same time, such as the Birth events. It presents attributes linked to events, such as sireId and matronId, but it is missing the linkage between these attributes and the artifacts. Another missing information is the states of the artifacts after each event is executed. This is due to the fact that the XES logging format is specified for activity-centric processes and thus there is a need for an artifact-centric logging format.

EventId	Activity	Timestamp	sireId	matronId	kittyId	cooldownPeriod	genes	auction
E1	Breeding	23/10/23 06:11:51	1475706			11225643		
E2	Breeding	23/10/23 06:11:51		1240424		11225643		
E3	Birth	24/10/23 10:12:36	1475706		1576916	0		
E4	Birth	24/10/23 10:12:36		1240424	1576916	0		
E5	Birth	24/10/23 10:12:36	1475706	1240424			8658320...	
E6	Auction Creation	25/10/23 15:22:96			1576916			
E7	Auction Creation	25/10/23 15:22:96			1542698			
E8	Auction Creation	25/10/23 15:22:96						Auct1
E9	Auction Creation	25/10/23 15:22:96						Auct2

Table 1.1: Cryptokitties XES log

1.3 Research Problem

Three challenges arise when trying to apply artifact-centric process mining to blockchain applications.

The first challenge is the choice of an adequate storage format for artifact-centric processes. The use of XES, as demonstrated in the literature [14, 15], will raise many issues when event logs resulting from the execution of artifact-centric processes are flattened. Flattening refers to the process of simplifying a multi-case notion event log into simpler, one-case notion logs by creating a classic event log for each object. It can also leads to **convergence and divergence problems**, as has been pointed out recently in the literature [16]. Because of these issues that arise when using XES to store artifact-centric event data, there is a need for a new logging format that aligns with the specifications of artifact-centric processes. At the start of our work, there were, to the best of our knowledge, only two object-centric logging formats in the literature: OCEL [17] and XOC [16]. However, OCEL does not support the storage of all the information related to artifact-centric event data. XOC supports most of that information but it does so by storing the entire relational model with each new even. This redundancy of information increases the size of the event log. Thus, a new format that addresses the limitations of XOC and OCEL is needed. Hence our first research question: *(RQ1) How to capture efficiently artifact-centric event data ?*

The second challenge is about the gathering of artifact-centric event data from blockchain applications. Blockchain's data structures and logging system, as intro-

duced in the context, were not designed with process awareness in mind. Indeed, the logs' structure does not fit any event data logging format. Several works in the literature proposed approaches to deal with this issue [18]. However, none of them considered the artifact-centric perspective and thus most of them used the XES format for storing event data collected from blockchain [19, 20, 21]. The identification and extraction of event data from blockchain needs to be done following the structure imposed by the storage format. Thus, there is a need for a new approach adapted to the requirements of the new format mentioned above. Hence the second research question of our thesis: *(RQ2) How to collect artifact-centric event data from blockchain applications ?*

The third challenge concerns the compatibility of artifact-centric logging formats with existing discovery techniques. Most existing artifact-centric discovery approaches are based on existing activity-centric discovery techniques and rely on XES logs but with some pre-processing steps and additional domain knowledge [8, 22, 23]. These approaches are not compatible with artifact-centric logs. Thus, there is a need for an artifact-centric discovery technique that takes as input an artifact-centric log, stored in the new format mentioned above, and that does not require pre-processing or domain knowledge. Hence, our third research question: *(RQ3) How to discover artifact-centric process models from artifact-centric event logs ?*

1.3.1 (RQ1) How to capture efficiently artifact-centric event data ?

It has been made clear in Sections 1.1 and 1.3 that XES is not suitable for storing artifact-centric event data because it introduce convergence and divergence problems. Convergence occurs when events linked to multiple instance of a an object are duplicated in its flatten log. This is illustrated by the duplication of Auction Creation events (E5 and E7) in the XES log of the motivating example (Table 1.1). Divergence, is a problem of creating a false causality between the events of instances of an object because the latter is linked to another object selected as the case notion of a log. This the case in the motivating example's XES log, where the case notion is that of a mother kitty and the log has consecutive events affecting its children (E6 and E7), each from a different instance. This can lead to considering that these events are linked to child kitty instance, when they actually refer to different instances, and thus represent a loop in the process. It has also been argumented that existing object-centric formats, such as OCEL or XOC, do not fully support all the characteristics of such data or do so in an inefficient and unscalable way. These

specification encompass the evolution of objects through lifecycles, the presence of an information model, and multiple interacting cases. In particular, OCEL does not support the evolution of objects nor relations. Without object evolution, the lifecycle of objects, as illustrated in the motivation example, cannot be captured. Although XOC supports these notions, it does so by storing the entire relational model with each new event. This redundancy of information increases the size of the event log. For optimization and adoption of the format, such redundancy needs to be avoided. Therefore, the proposition of a new format should be investigated.

The first shortcoming of existing object-centric formats, when it comes to artifact-centric event logs, is capturing object evolution. Indeed, these formats link events to the object they affect but without any indication of the nature of the change. We define object evolution as the changes made to an object's attributes by business operations (activities/tasks) and the state [3] the object reaches after a business operation. Thus, the problem of capturing object evolution becomes a problem of storing attribute changes and object states. However, since the attribute changes are made by activities, they should be linked to events as well as to objects. Hence, the first sub-question: *(RQ1.1) How to store attribute changes and object states, and link them to objects and events without redundancy ?*

The second shortcoming is about capturing the interactions between objects. Indeed, objects can interact with objects of different or same types. This is not supported in OCEL and in the case of XOC, in which relations are stored, this is done by duplicating all relations with each new event. Thus, the potential solution to this problem should focus on optimization to avoid redundancy and oversized logs. Additionally, the evolution of relations, i.e., creation, update, and end, is not explicitly supported in the other formats, yet it has the potential to indicate precisely the behavioral dependencies between events and by extension between objects' lifecycles.

Hence, the second sub-question: *(RQ1.2) How to store relations and capture their evolution without redundancy ?*

1.3.2 (RQ2) How to collect artifact-centric event data from blockchain applications ?

Blockchain data is not structured in an event data friendly manner, i.e., the structures used to store data about the execution of smart contracts does match that

[3] In the context of artifact-centric processes, the "state" of an object refers to its current condition or status within the lifecycle of a business process. For example, in a procurement process, the states of a purchase order artifact could include "created", "approved", "fulfilled", and "closed".

of event logs used in BPM. Additionally, blockchain applications' data and smart contract event logs are scattered across blocks and might not be linked to a process instance identifier. This fragmentation of process data requires an approach to map blockchain data to event data. For instance, an application may have three smart contracts but the execution of transactions generates logs from only two of them because the other smart contract does not have a logging functionality. Furthermore, blockchain data and logs are stored in a cryptographic format which requires decoding. Finally, a smart contract's code does not necessarily include explicit object types to represent the business artifact. In the case of the motivating example, if we look at the smart contracts' code, we see that the auction artifact is not explicitly an object type, but its attributes are present in the code and are manipulated by the smart contracts' functions. Thus it requires efforts to identify and collect process execution data. In the literature, the attempts to solve this problem were solely focused on activity-centric event data. The approaches to extract object-centric event data like OCEL or XOC were specific to relational databases. Relational databases are structured, whereas blockchain data is unstructured. This calls for the investigation of a new approach tailored for extracting artifact-centric event data from blockchain.

Thus, the new approach needs to also include the reconstruction of the information model from the smart contract code.

The first challenge when it comes to identifying artifact-centric process data from blockchain is the identification of artifact structures. In the approaches specific to relational databases, this was straightforward as the artifact simply referred to tables. In blockchain data artifacts are not explicitly defined. Hence, the first sub-question : *(RQ2.1) How to identify artifact structures from blockchain data ?*

The second challenge is about the identification of the information on artifacts that allows to discover their relations. In relational databases, the relations between artifacts can be identified using primary and foreign keys. In blockchain, similarly to the artifact structures, the interactions between them are implicit and they need to be inferred. Hence, the second sub-question: *(RQ2.2) How to identify artifact relations from blockchain data ?*

Once the artifacts and their relations are identified, they need to be stored in an artifact-centric event log along with their linked events. Hence, the third sub-question: *(RQ2.3) How to generate artifact-centric event logs from event data collected from blockchain ?*

1.3.3 (RQ3) How to discover artifact-centric process models from artifact-centric event logs ?

Proposing a new artifact-centric logging format raises the challenge of adapting it to process mining techniques. In the cases of OCEL and XOC new discovery techniques and process representations were proposed [15, 24]. However, these the discovery techniques do not consider object evolution and data conditions which drive the execution of artifact-centric processes and lead to behavioral dependencies between artifacts. Consequently, they discover flat models with no data conditions, such as the process model of the motivating example in Figure 1.2 where we see no hierarchical structure of activities and no data conditions. Hence, the representation do not depict artifacts' lifecycles and only show interactions between artifacts on a high level. Another feature of artifact-centric process models that is not considered by these approaches is the hierarchical structure of activities, i.e., which group of activities when executed together help in the achievement of a business goal.

To the best of our knowledge, no approach discovers artifact-centric process models from object or artifact-centric logs. Some approaches use mechanisms to transform discovered Petri nets into artifact-centric models [23], but the discovery is done from pre-processed XES logs using existing activity-centric discovery techniques. Thus, to discover artifact-centric process models from artifact-centric logs we ask the following sub-question: *(RQ3.1) How to discover an information model from an artifact-centric event log ? (RQ3.2) How to discover data conditions from an artifact-centric event log ? (RQ3.3) How to discover the hierarchical structure of activities from an artifact-centric event log ?*

1.4 Thesis Objectives and Contributions

Considering the aforementioned research questions, we outline the core objectives of this thesis and the contributions that have been proposed to fulfill them in the forthcoming sections.

1.4.1 Thesis Objectives

Given the research challenges outlined earlier, this thesis aims to achieve the following primary objectives:

Objective 1 Propose an artifact-centric logging format that supports object evolution by storing attribute changes, and relations and their evolution (c.f., RQ1.1 and RQ1.2):

- Link attribute changes and relations to objects and events;
- Avoid redundancy by design;

Objective 2 Identify the elements of artifact-centric data from blockchain data (c.f., RQ2.1 and RQ2.2):

- Identify artifact structure
- Identify artifact relations

Objective 3 Propose a mapping from blockchain data to artifact-centric event data elements(c.f., RQ2.3);

Objective 4 Automate the collection of artifact-centric event data from blockchain data and the generation of event logs from these event data (c.f., RQ2.3);

Objective 5 Automate the discovery of artifact-centric process models from the generated artifact-centric event logs (c.f., RQ3.1, RQ3.2 and RQ3.3):

- Automate the discovery of information models
- Automate the discovery of data conditions
- Automate the discovery of the hierarchical structure of activities

It is noteworthy that the proposed work in this thesis needs to be: (i) validated through proof of concepts, and (ii) evaluated through different experiments on public blockchain application logs to allow comparison with related approaches. Furthermore, the implementation, experiments, and results should be detailed.

1.4.2 Thesis Contributions

To address the research challenges described above and reach the objectives of this thesis, we propose the following contributions, illustrated in Figure 1.3 :

1. **An artifact-centric logging format**

   To achieve Objective 1, we propose a logging format tailored to capture all the characteristics of artifact-centric process data. This format supports object evolution and relations and introduces the concept of relation evolution to bring more precision to the identification of behavioral dependencies. Furthermore, it supports all cardinalities of relations (one to one, one to many and many to many) and allow for the capturing through relations of reflexive

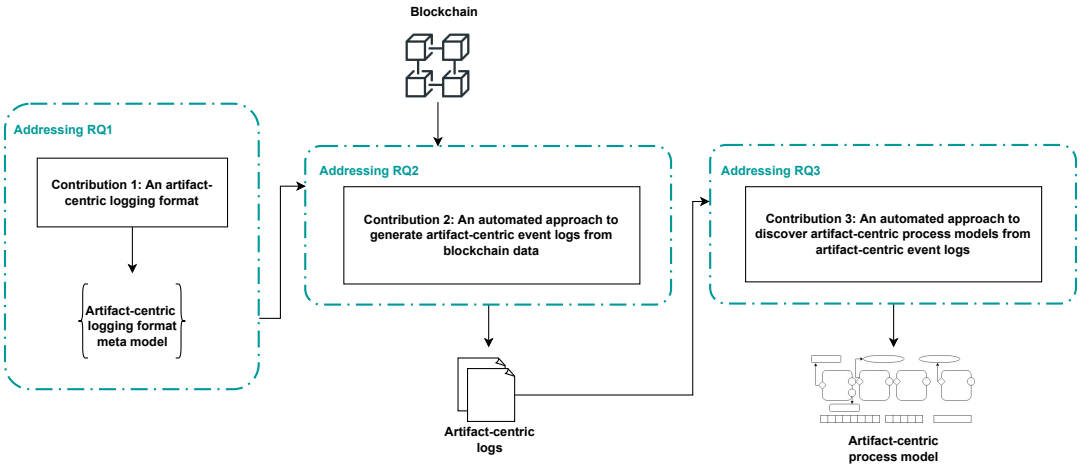

Figure 1.3: Our contributions in regard to our thesis's research problem

interactions, i.e., interactions between objects of the same type. The format's structure is designed so as to avoid redundancy and thus optimize the log size.

2. **An automated approach to extract artifact-centric event data from blockchain and generate artifact-centric event logs**

   To achieve Objectives 2, 3 and 4, we propose an approach to convert blockchain data into artifact-centric event logs using domain knowledge. The domain knowledge is provided by domain experts and consists of information on the smart contracts of the blockchain application and the business entities interacting in that application. To achieve this result we follow the following steps :

   (a) Extracting artifact-centric data from blockchain (Objective 2): Using the domain knowledge, we query the blockchain logs to collect the required data;

   (b) Mapping blockchain data into artifact-centric event data (Objective 3): we rely on mapping rules, built based on domain knowledge, to determine which raw data element corresponds to which artifact-centric event data element;

   (c) Generating structure artifact-centric event logs (Objective 4): we structure the previously identified artifact-centric elements in a log according to the specifications of our proposed logging format.

3. **An automated approach for the discovery of artifact-centric process models from artifact-centric event logs** To achieve Objective 5, we present a novel process mining technique which does not require domain knowledge and

takes as input an artifact-centric event log, in the format previously proposed, and discovers a artifact-centric process model by combining the following automated solutions :

(a) Discovering data conditions : we focus on determining which data conditions are always verified before an activity by analysing the attribute changes. This allows us to group activities which have common data conditions into one phase ;

(b) Discovering hierarchical structure of activities : Similarly to the grouping of activities into phases, our algorithm groups phases with common condition to get another level of abstraction. Our technique takes into consideration the necessity to have a coupling which allows for parallelism while avoiding that different activities in different phases can be triggered by the same data condition;

(c) Discovering artifact interactions : We discover interactions between artifacts of different types as well as between artifacts of the same type (reflexive interaction) by relying on data conditions. This means that when a data condition relating to one artifact triggers an activity in the lifecycle of another artifact, it is a proof of the behavioral dependency between both artifacts.

We publicly provide our implemented tools and experimental results to allow for reproducibility and comparisons with related studies, especially those who use the same case study.

1.5 Thesis Outline

The thesis is outlined as follows. The Chapter 2 presents the concepts related to business process management, artifact-centric process mining, and blockchain. The chapter 3 presents the related work. Our proposed format for storing artifact-centric data is presented in Chapter 4. Chapter 5 details our approach to extract artifact-centric event data from blockchain applications and transform them into logs in our proposed format. In Chapter 6, we present our technique to discover artifact-centric process models from logs in the newly proposed format. Finally, we conclude on our work and present future research directions in Chapter 7.

Chapter 2

Background

Contents

The purpose of this chapter is to provide a basic understanding of the main concepts this thesis is founded on. We present the different approaches to process modeling in Section 2.1. Section 2.2, introduces artifact-centric process mining. In Section 2.3, we present the technology of Blockchain and how it is used for BPM.

2.1 Business Process Modeling

In the evolving landscape of BPM, the distinction between artifact-centric and activity-centric process modeling has emerged as a fundamental conceptual divergence, influencing how organizations design, implement, and analyze their business processes. This differentiation is pivotal in understanding the shift from traditional, rigid process architectures to more flexible paradigms that better accommodate the data-driven and dynamic nature of contemporary information-rich business environments [6]. While the activity-centric approach to process modeling offers clarity and control for processes with a known sequence of tasks, the artifact-centric approach provides the flexibility required to manage processes characterized by variability and the need for responsiveness to dynamic business conditions.

The artifact-centric approach emerged along side the object-centric approach which also aims to bring a data perspective to process models. Both these terms are used in the literature [14, 8] and sometimes refer to the same group of techniques. However, in our opinion the difference between them lies in the nature of the data entities being manipulated in the processes as well and their role in the execution of the processes. Specifically, we consider in this work that object-centric processes involve objects that do not evolve, i.e., their properties do not change with each event. We also consider that object-centric processes are not driven by the objects as their states do no change.

Objects are similar to artifacts in that they represent entities within a process. However, the term "object" is more inclusive and can refer to any entity (physical, digital, conceptual) involved in the process, not limited to primary business entities. In our work, we focus on artifact-centric processes but we may use the term object instead of artifact when the context requires it.

In the following we will detail the principles of both activity-centric and artifact-centric approaches and provide as an example a modeling language that follows each one of them.

2.1.1 Activity-centric process modeling

Activity-centric processes consist of a flow of tasks or activities centered around the ordered execution of business operations. Activity-centric process modeling emphasizes a structured sequence of tasks or activities as the core of a process model. In this traditional BPM approach, the models are delineated by the orderly execution of these activities, ranging from elementary tasks to complex sub-processes. Activities serve as the foundational unit of modularization in this approach, i.e., the

basic building block used to construct or describe a process model. This approach is particularly effective for processes characterized by predictable flows and minimal variations, with the flow of the process being guided by transitions that illustrate the movement from one task to the next.

2.1.1.1 Business Process Model and Notation

Business Process Model and Notation (BPMN) [12] is the defacto standard for business process modeling in both academic research and industry practice. It is also an activity-centric process modeling with a standardized graphical notation. It allows a detailed depiction of process flows using a sequence of activities, decision points, parallel paths, and synchronization points. BPMN diagrams consist of flow elements (events, activities, gateways) connected by sequence flows, thereby illustrating the order of operations within a process.

Events an event is an action that happens automatically as part of a process and can start, intermediate, or end a process. They are represented by circles and can be further divided into types like Start, End, and Intermediate (e.g., timer events, message events).

Activities any work or business operation that is performed within a process can be represented by an activity. Activities can be tasks (simple activities) or subprocesses (complex activities that are processes themselves). They are depicted as rounded rectangles.

Gateways controlling the divergence and convergence of sequence flows in a process is done through gateways. They represent decision points that can affect the path of a process. They are shown as diamond shapes and serve different purposes for the execution of the process depending on their symbol. For example, the exclusive gateway, represented by diamond shape with an "X" inside, determines a path among two or more alternatives based on a condition or decision, ensuring that only one path is taken.

Sequence flows the order of activities within a business process is defined using sequence flows. They are graphical elements represented by solid lines with an arrowhead, connecting flow elements in a sequence diagram. The direction of the arrow indicates the flow of the process, from one element to the next, illustrating how operations are executed in a prescribed order.

The elements of the BPMN graphic notation were used to illustrate the motivating example in Section 1.2

2.1.2 Artifact-centric process modeling

Artifact-centric processes are a type of business processes where the primary focus is on the manipulation and progression of business entities, known as artifacts. Artifacts are business relevant entities like documents, files, records, objects, or any other form of data that holds significance within the process. Additionally, artifacts in such processes often transition through various states or stages indicating their status or progress as they move through the work flow. The central element of an artifact-centric process is the data artifact itself. Changes in the states of artifacts can trigger different paths or actions within the process.

Contrasting with the activity-centric approach, artifact-centric process modeling describes processes using artifacts, that are manipulated and transformed throughout the process lifecycle. This paradigm emphasizes the state changes of artifacts, caused by the execution of activities, to dictate the flow of the process. Artifact-centric modeling languages structure process according to two key concepts :

Information model an artifact's information model refers to the structured representation of the data or information contained within the artifact. It defines the attributes, properties, metadata, and relationships associated with the artifact. Essentially, it outlines what data an artifact can store and how that data is organized.

Lifecycle an artifact's lifecycle represents the various stages or states through which the artifact progresses during the course of the process. It defines the sequence of events or changes that an artifact undergoes, from its creation to its eventual disposition or completion. It can be seen as a micro-process contains all activities / business operations that can affect an artifact.

2.1.2.1 Guard Stage Milestone

Guard Stage Milestone (GSM) is one of the most used artifact-centric modeling languages. It delineates the progression of an artifact's lifecycle by using guards, stages, and milestones, while characterizing the information model through data and state attributes [25].

Guards a guard is a condition or a rule that determines whether a particular transition or action in a process should take place. They are used to specify under what circumstances a particular task or event can occur.

Stages a stage represents a specific phase or step in a process or workflow. They provide a way to organize and structure a process into manageable units.

Stages are used to group related activities or tasks together, often with a common objective or outcome. They are opened and can be executed when their guards' conditions evaluate to true.

Milestones a milestone is a significant point or event within a process that marks the achievement of a specific goal or the completion of a critical phase. Milestones are used to track progress and can indicate key moments in a project or workflow. They mark the closing of a stage.

In the motivating example 1.2, the auction phase can be considered as a GSM stage. When a user wishes to sell their kitty, they initiate the `CreateAuction` stage, which generates a new auction in the Created state. Other users can then place bids to purchase the auctioned kitty. Upon the acceptance of a satisfactory bid, the `CompleteAuction` stage is triggered, leading to the transfer of the kitty to the new owner and marking the auction as Successful. Users who initiated the auction can also cancel it by invoking the `CancelAuction` stage, resulting in the auction being marked as Cancelled.

Similarly, the procreation phase can be considered as a GSM stage. After the breeding process, both the mother and the father enter specific states: the mother is in the Pregnant state, and the father is in the FutureFather state. Following a predetermined cooldown period, users can initiate the `Birth` stage, leading to the birth of a new kitty, which is then in the Born state. After this event, the mother and the father transition to the BecameMother and BecameFather states, respectively.

In Fig. 2.1, we provide an excerpt of the representation of both a kitty's and an auction's lifecycles using the GSM modeling language. These artifacts, namely kitty and auction, possess data attributes that constitute their information models, such as `tokenId`, `cooldownPeriod` and `startingPrice`. These attributes also indicate the existence of relations between artifacts through the referencing of other artifacts, e.g., `SiringWithId` and `KittyId`. Moreover, these artifacts posses status attributes, such as Pregnant for the kitty artifact and Successful in the case of auction artifact. Fig. 2.1 also illustrates the potential nesting of stages, exemplified by the Procreation and SaleAuction stages. The Procreation stage, for instance, includes a guard with a sentry comprising a data condition (`k.'cooldown'`$\leq$`currentTime`), and a milestone (BecameMother) with a sentry comprising an internal event related to the kitty's milestone (`k.'Pregnant'.achieved()`). When the guard evaluates to true, the Procreation stage is initiated, enabling the activation of its sub-stages, Breeding and Birth, when their respective guards also evaluate to true. For example, the sentry of the Birth stage's guard includes an external event (`k.'giveBirth'.onEvent()`), to be triggered by a user, and a data condition (`k.'Pregnant'`).

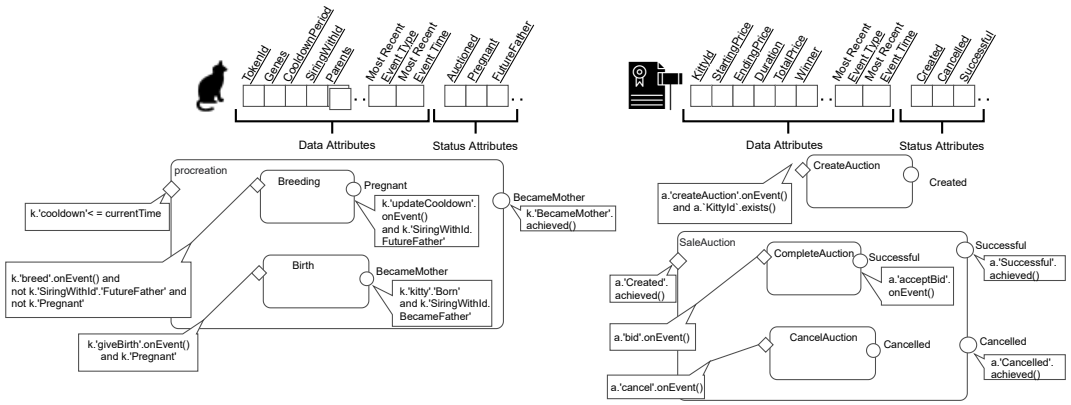

Figure 2.1: GSM process model associated with the Cryptokitties example

Artifact-centric modelling offers greater flexibility than activity-centric modelling by adapting the process flow based on the current state of artifacts, making it more suited to dynamic environments where process paths may need to vary based on contextual data.

2.2 Process Mining

Process mining [26] is a discipline at the intersection of data science and process analysis. It combines the data-centric analysis with the model-based process analysis. It provides a set of techniques and tools which provide insights that allow for a deeper analysis of process performance and conformance. It is complementary to the monitoring phase and essential for the discovery phase. Furthermore, it can help in delays prediction, decision making, and process redesign. Event data provided by Business Process Management Systems or Process Aware Information Systems is the basis for process mining. This data is structured in event logs according to a logging format, such as XES [13], and used as input for the process mining techniques. Process mining techniques have been broadly categorized into four main areas: Process Discovery, Conformance Checking, Performance Mining, and Variants Analysis. Each category offers a unique lens through which organizations can analyze and refine their processes, leveraging the rich data available in event logs to drive operational efficiency, compliance, and innovation [27, 5]. In the following we will provide a description of XES, the most used format to store event data for process mining, and we will present the four categories of process mining techniques.

2.2.1 Process Discovery

Process Discovery [28] is the first and perhaps most foundational category of process mining techniques. It focuses on extracting an accurate process model from event logs and usually without any additional input. This category of techniques uses algorithms to identify patterns and sequences in the event data, constructing a comprehensive model that reflects the actual execution of the process. The discovery of process models can reveal the underlying structure of business processes, offering insights into how tasks are organized and executed.

2.2.2 Conformance Checking

Conformance Checking techniques [29] compare the observed behavior recorded in event logs against predefined process models to identify deviations, non-compliance, and areas of improvement. This type of techniques is crucial for verifying whether the actual execution of processes aligns with the intended process design. They help in pinpointing discrepancies between a model and a real process executions, facilitating targeted corrective actions to enhance process compliance. Metrics such as fitness, precision, and generalization play a key role in evaluating the degree of conformance.

2.2.3 Performance Mining

Performance Mining [30] extends beyond the structural aspects of process models to analyze the performance characteristics of processes. This category of techniques focuses on extracting insights related to time, cost, and resource utilization from event logs. Performance mining techniques can identify bottlenecks, evaluate the impact of different process paths on performance, and suggest optimizations. By analyzing durations between events, workload distribution, and service levels, performance mining offers a detailed understanding of process efficiency and effectiveness, enabling data-driven decision-making to improve process outcomes.

2.2.4 Variants Analysis

Variants Analysis [31] delves into the differences between various instances of a process, focusing on understanding and managing process variability. This category of techniques is particularly valuable in environments where processes are subject to frequent changes or where customization is common. By comparing different variants of a process, organizations can identify best practices, understand the reasons

behind variations, and standardize processes where beneficial. Variants analysis can reveal insights into how and why certain process paths lead to better outcomes, facilitating the replication of success across the organization.

2.2.5 eXtensible Event Stream

Standardized data representation is paramount for process mining. The eXtensible Event Stream (XES) [13] logging format is a standard that emerged in 2010 as a solution to this need, providing a uniform framework for storing, managing, and analyzing process-related data. It allows for the interoperability between process mining tools and applications, ensuring that event data from various sources can be easily shared, understood, and analyzed across different platforms.

At its core, the XES standard is designed to be both flexible and extensible, capable of accommodating the wide variety of event data encountered in process mining. An XES log file is structured hierarchically, comprising three primary levels: log, trace, and event.

Log: The top-level element that acts as a container for multiple traces. A log represents a collection of process instances, often corresponding to the execution of a particular process within an organization.

Trace: Each trace within a log corresponds to an instance of the process, i.e., one process execution, encapsulating the sequence of events that occurred during that instance. Traces allow for the aggregation of events that collectively describe a single process execution.

Event: Events represent the lowest granularity level of the XES model. They represent individual executions of activities within a process instance. Each event is characterized by a set of standard attributes, such as timestamp, resource identifier, and activity name, along with any number of custom attributes that can capture additional details relevant to the specific process context.

This format was used to represent the event data of the motivating example in Table 1.1 (Section 1.2).

2.3 Blockchain

Blockchain, the technology behind cryptocurrencies, gained public recognition in 2009 with the launch of Bitcoin. This innovation did not arise in isolation but

rather from integrating several pre-existing ideas, combining cryptographic security, decentralized networks, proof-of-work, and innovative mechanisms to prevent double-spending. Before 2009, these ideas existed in various academic and technical discussions but did not capture widespread attention. Bitcoin's successful implementation demonstrated the practical utility of blockchain, initially for conducting financial transactions in low-trust environments without a central authority. Soon after, the second generation of blockchain platforms expanded its applicability beyond financial use cases.

Blockchain technology [4] is characterized by its decentralized nature. It offers a transparent, secure, and immutable framework for storing and transferring data and assets. Across a network of nodes, blockchain maintains a consistent, replicated ledger, eliminating centralized control. Some blockchain platforms enhance their capabilities with smart contracts, i.e., executable programs that automate and enforce contract terms based on predefined conditions. Blockchain's architecture is a series of interlinked blocks, where each block acts as a container of verified transactions. These transactions are authenticated and agreed upon by nodes in a peer-to-peer network. The chain's integrity is guaranteed by the fact that altering a single block would invalidate the entire chain. In addition, its immutable nature renders it a reliable source for data analytics.

The following sections delve deeper into blockchain's data recording structures Section(2.3.1), examine the execution and event logs of smart contracts (Section 2.3.2), and give an overview of the usage of Blockchain for BPM (Section 2.3.3).

2.3.1 Transactions and Logs

Blockchain transactions [32] are the fundamental records stored within the blockchain. They can range from financial exchanges like cryptocurrency transfers to the documentation of asset ownership. The process of adding new record to a blockchain begins with the submission of a transaction to the blockchain network, where it undergoes verification and validation by nodes through an established consensus mechanism. Once validated, the transaction is incorporated into a new block alongside other confirmed transactions. This block is then chronologically added to the blockchain, cementing the transaction into a permanent and unalterable record.

Blockchain logs, in contrast, are chronological records of events within the blockchain network. These logs provide detailed accounts of activities like the addition of new blocks, submission and confirmation of transactions, or alterations in network protocols and settings. Both transactions and logs are pivotal to the blockchain ecosystem, offering a transparent and tamper-proof record of network activities.

2.3.2 Smart Contract Execution and Events

Smart contracts [33] are self-operating digital contracts recorded on the blockchain. These contracts are essentially programs that autonomously enact the terms of an agreement upon the fulfillment of predefined conditions. Their applications are diverse, ranging from automating financial transactions to streamlining supply chain processes, ensuring adherence to contractual agreements without manual intervention.

In a blockchain, smart contracts exist as code and are autonomously executed when specific conditions are met. For instance, a smart contract might be programmed to automatically execute a payment transfer between parties once a delivery condition is met. This automation of contractual obligations underpins the efficiency and reliability of blockchain-based transactions.

Events within smart contracts serve as critical notifications triggered by certain actions or stipulated conditions in the contract's code. These events can inform parties about the fulfillment of specific conditions or instigate subsequent actions within the contract framework. For instance, in the realm of supply chain management, an event in a smart contract could be activated upon the receipt of goods at a specified location, thereby triggering an automatic payment release to the supplier or initiating the next contractual phase in the supply chain process.

2.3.3 Blockchain and BPM

By leveraging blockchain, businesses can achieve unprecedented levels of process security, and trust by ensuring the non-repudiation of process transactions and activities [34]. Blockchain's inherent transparency fosters trust among process participants. In BPM contexts, this means every action taken within a process is recorded on the blockchain, visible to all authorized stakeholders, and immutable once recorded. This transparency is instrumental in sectors where traceability and accountability are paramount, such as supply chain management, where blockchain can track the provenance and status of goods in real-time. Smart contracts are among the most significant blockchain innovations for BPM. They enable automated, conditional execution of process steps, ensuring that processes are carried out exactly as predefined. For instance, in financial services, smart contracts can automate payments and settlements, triggering transactions only when agreed-upon conditions are met, thereby reducing manual oversight and error.

Smart contracts can be used to exchange messages between participants on the state of the process or to automate certain activities. Furthermore, they can be

used as part of a complete blockchain-based business process management system (BBPMS). These BBPMS run on top a blockchain and allow their users to design a process model, generate its corresponding smart contracts, communicate with the blockchain to execute tasks and track the state of process instances. The state of the process instances is maintained on the blockchain using smart contract events, and the workflow is managed by one or many smart contracts which act as entry points.

2.4 Conclusion

In this Chapter, we first outlined the difference between artifact-centric and activity-centric approaches and illustrated them with examples. Then, we introduced the discipline of process mining and the different categories of process mining techniques. Finally, we introduced the technology of Blockchain and its underlying concepts. The data resulting from the execution of artifact-centric processes is the focus of Chapter 4, where our contribution involves the adequate storage of this execution data. In Chapter 5, we use concepts from artifact-centric process and blockchain in our contribution which converts blockchain data into artifact-centric event data. We use the concepts of process mining, in particular artifact-centric process mining, to propose a discovery approach that use artifact-centric event data and produces GSM models.

Chapter 3

Related Work

Contents

3.1 Introduction

In this chapter, we examine existing works that pertain to different aspects of artifact-centric process mining in the context of blockchain in order to position our work within the literature. We categorize these works into three distinct groups, with each group corresponding to one of the main research questions introduced in the context section: artifact-centric logging formats (c.f., RQ1), blockchain logging for process mining (c.f., RQ2), and artifact-centric process discovery (c.f., RQ3). The first group, presented in Section 3.2, comprises the existing approaches related to the topic of solving the issue of artifact-centric event data storage. The second group, reviewed in Section 3.3, show cases the different works that consider the use of blockchain data for the purpose of process mining. The last group, presented in Section 3.4, encompasses the different approaches which aim to discover artifact-centric process models. We thoroughly analyse these works to highlight the challenges we aim to address in this thesis and compare our work to them.

The work in this chapter was published in conference proceedings [18, 35, 36] and peer-reviewed journal [37].

3.2 On storing artifact-centric event data

In this section we present the current artifact or object-centric formats proposed in the literature. We discuss the limitations of each format when it comes to addressing the challenges raised through the sub-questions RQ1.1 and RQ1.2. Due to the novelty of the topic, we could only identify three related works.

Li et al. [16] introduce XOC (eXtensible Object-centric logs), a novel object-centric logging format that extends the XES (eXtensible Event Stream) standard. Distinctly, XOC avoids the traditional reliance on a predefined case notion inherent in XES, enabling the accommodation and handling of multi-dimensional data without necessitating its flattening. This format is specifically tailored for object-centric data from systems such as Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM), focusing on how events impact the state of an information system's relational database. XOC's primary objective is to chronicle the evolution of a database in tandem with the events occurring within the information system.

Adopting an object-centric perspective, XOC facilitates the logging of multiple case notions, to avoid the convergence and divergence issues encountered when transforming object-centric logs into activity-centric formats like XES. In such object-centric systems, data is organized around entities, or 'objects', such as records in database tables. These objects represent data elements, with classes grouping similar objects together (for example, all 'order' records constituting the 'order' class). Inter-class relations, indicative of dependencies between database tables, are articulated through foreign and primary key references.

XOC encapsulates an object model that mirrors the state of the relational database at a given moment. This model comprises objects, their relations, classes, and object attributes. For instance, events, which correspond to changes in database records, are captured in redo logs and can be derived either by aggregating simultaneous changes from these logs or via domain-specific knowledge.

The methodology proposed by Li et al. [16] for generating XOC logs emphasizes the use of relational databases as a primary source of event data. It considers that the tables of a relational database represent objects and delineates a process for converting redo logs [1] [16] into events and employing foreign keys to deduce

[1] Redo logs are tables containing a list of all changes made to a database

relationships between these objects. Consequently, XOC logs offer a comprehensive view of the database's evolution and its associated events, encapsulating the dynamic state alterations within the database. Each log entry in XOC includes not only the event attributes but also references to related objects and a snapshot of the database post-event occurrence. However, this approach of storing the entire relational model data for every new event, instead of just the new objects or relations, renders the XOC format resource-intensive in terms of file size and structurally suboptimal.

Another object-centric event logging format is introduced by Ghahfarokhi et al. [17]. The propose format, OCEL, aims to address the limitations of traditional logging format, e.g., XES, by allowing events to relate to multiple objects. The key feature of OCEL is its ability to associate each event with different objects, overcoming the convergence and divergence problems found in single-case oriented logs. It supports the storage of events, objects, and their attributes. Each event stores information about the execution of an underlying business process activity, e.g., breeding a kitty (Section 1.2), and the objects affected by the executed activity. Objects represent physical and informational entities relevant to business processes, e.g., a kitty, an auction (Section 1.2). In an OCEL log, events and objects are uniquely identified and may have several attributes (attributes are properties of OCEL elements, e.g., timestamp is an attribute of events). Each object is associated with an object type and multiple objects can be linked to a single event (i.e. the execution of an activity may affect multiple objects).

In the official standard documentation, the authors summarize the concepts of OCEL Logs in a class diagram, as illustrated in Fig. 3.1. The main classes considered and the relations linking them are as follows:

- A **Log** consists of **Events** and **Objects**.
- An **Event** consists of the following mandatory attributes: an **Event Id**, an **Activity** and a **Timestamp**. It can have additional **Attributes**.
- Each **Attribute** has a **Name** and an **Attribute Value**.
- An **event** relates to one or more **Objects**, which it affects.
- An **Object** contains an **Object Identifier**, an **Object Type**, and zero to many **Attributes**. It can also be related to one or many **Events**.

OCEL presents an efficient storage of object attributes, by decoupling attributes at the object level from the event level, as shown in Figure 3.1, it avoid replicating this information with each event affecting the object. It also supports lists and

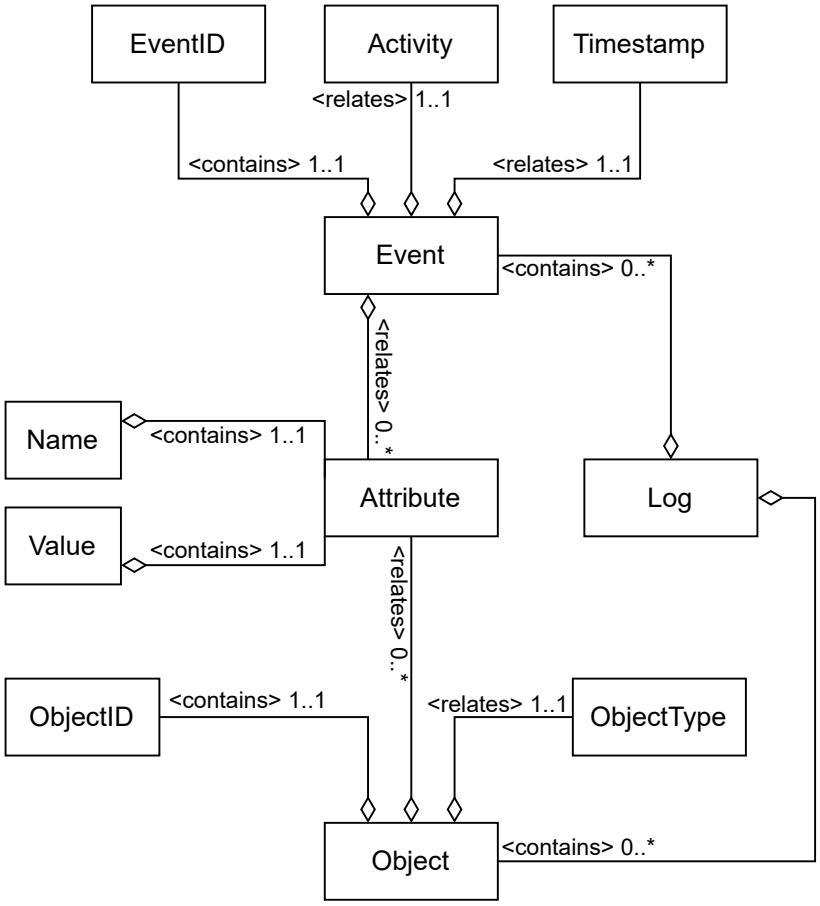

Figure 3.1: OCEL model

maps elements as data structures. Nevertheless, it is essential to highlight that OCEL does not encompass the capacity to store relations between objects, and it does not capture the changes that objects may undergo over time. Therefore, it does not allow to capture all the information present in artifact-centric event data.

Goossens et al. [38] propose an extension to the OCEL format that supports object evolution. They argue that there is a need for comprehensive event and object attribute storage in logs, including attributes with dynamic values and the ability to unambiguously link attributes to objects and events. Therefore they introduce a format called Data-aware Object-Centric Event Log (DOCEL) and an associated translation algorithm to convert XES logs into DOCEL logs. However, DOCEL does not support the storage of object relations. Additionally, the dynamic attributes are stored in dynamic tables where the identifiers of the events and objects linked to them are duplicated. The algorithm was not implemented nor tested on real data, thus we cannot evaluate its feasibility.

In Table 3.1, we summarize our comparison of the previously presented logging formats. We evaluate to which extent they fulfill the requirements to answer RQ1.1 (How to store attribute changes and object states, and link them to objects and events without redundancy?) and RQ1.2 (How to store relations and capture their evolution without redundancy?), we use as criteria conciseness (avoidance of redundancy) and the required artifact-centric concepts that need to be supported. We use the notation "x" to express that a criterion is fulfilled, "/" to express that is partially fulfilled and no notation when it is not fulfilled.

Format	Object changes	Relations	Relation evolution	Conciseness
XOC [16]	/	**x**		
OCEL [17]				**x**
DOCEL [38]	/			/

Table 3.1: Comparison of object-centric logging formats

Synthesis All the related work that deal with the storage of artifact-centric or object-centric event data aim to overcome the limitations of classic logging formats such as XES. The limitations of classic formats are apparent when dealing with process event data with multiple case notions which they do not support. The issues of classic formats extend to process mining where problematic logs presenting convergence and divergence lead to erroneous process mining results. The related work deal with these limitations by allowing the association of events with multiple objects. They all store the attributes of events and objects but only XOC stores the information related to relations between objects which is a crucial notion in artifact-centric event data. Furthermore, they differ in the structure of their logs,

some being more efficient than others. In particular, we found that XOC is inefficient in term of storage because of the data duplication and similarly DOCEL presents some data duplication. In our novel format we focus on supporting all aspects of artifact-centric event data in an efficient way. We aim to propose an optimised strucuture for artifact-centric event log where there is no redundancy. We follow the structure of OCEL as it is most efficient, and we enhance it with the rest of the concepts from artifact-centric event data, i.e., object evolution and relations. Hence, our format captures all the information present in artifact-centric event data but without redundancy.

3.3 On extracting event data from blockchain

Blockchain programmable platforms has brought considerable advancements in collaborative business processes. One of the anticipated benefits was the provision of auditable traces for business process execution. However, practical challenges arose, particularly concerning the structure of blockchain logs, which proved to be inadequate for process mining techniques.

We identify two main categories of approaches, each dealing with the inadequacy of blockchain logs for process mining at different stages of the blockchain data lifecycle. We define the blockchain data life cycle as: the *entrance*, i.e., when data is sent to the blockchain, *the storage*, i.e., when the received data is stored, in general via smart contracts, into the blockchain, and the *retrival*, i.e., when the data is queried or fetched from the blockchain.

Pre-blockchain Approaches: These tackle the problem at the source, handling the logging of event data into the blockchain. They aim to prevent the issue by structuring event logs before storing them on the blockchain. The event data is this category is generated by information systems outside of blockchain and the works of this category focus on the storage of this data on the blockchain.

Post-blockchain Approaches: These focus on transforming data already found in blockchain, i.e., resulting from the execution of smart contracts, into process mining suitable formats. This involves extracting data from the blockchain and converting it into a format that is amenable to process mining techniques

In the following we present the related works on the topic of extracting event data from blockchain using the above categories.

3.3.1 Pre-blockchain Approaches

In their study, Ekici et al. [39] address the issue of blockchain event data inadequacy for process mining by implementing a system that processes and stores event data as objects on the Hyperledger blockchain. This data originates from business processes executed by an external BPMS. They developed an application that sends event data to a smart contract for validation and storage on the blockchain, where only the caseID and activityName are required for logging, as timestamps are appended during the validation process.

Brinckman et al. [40] tackle the problem of log creation and storage in the blockchain, focusing on scientific processes. They use the Pegasus workflow tool, extended to transmit log events to a private Ethereum blockchain via a smart contract. They also explore converting blockchain data into a relational database format for SQL querying and apply machine learning algorithms to public Ethereum transaction data for detecting unusual account activities.

Zimina et al. [41] address challenges in educational processes by leveraging blockchain for data storage, particularly for reporting and course completion rates. They collect logs from a Moodle-based system, storing learning process events as smart contracts on the blockchain. These contracts, containing event descriptions and parameters, record events with timestamps and student addresses. The data is then exported for creating student models or educational reports.

Engelenburg et al. [42] aim to develop a blockchain-based system for secure information sharing between government and businesses, protecting sensitive data. The system stores events and information sharing rules on a blockchain, ensuring confidentiality based on company policies. The blockchain is designed to add each new event as a block, with consensus achieved through dual-party confirmation and network node verification.

Tonnissen et al. [43] investigate a blockchain solution to address media breaks in large business processes, which lead to incomplete event logs. They propose using blockchain as a connector between process participants in a multinational company's supply chain, recording data records from suppliers' IT systems to the blockchain, and forwarding them to the receiving company's IT systems through smart contracts. This integration aims to facilitate end-to-end process mining.

3.3.2 Post-blockchain Approaches

Di Ciccio et al. [44] propose a method for tracing collaborative Business Processes (BPs) on the Ethereum blockchain, utilizing Caterpillar [45], a blockchain-based

BPMS. This system translates a BP model into two types of smart contracts: one for processing (process factory) and another for managing and executing activities (worklist factory). The worklist smart contract, acting as the gateway for activities, is used as the identifier for process instances. To track a process instance, the method involves filtering transactions sent to the worklist smart contract's address, with each transaction linked to an activity. The attributes of these activities, such as identifiers and parameters, are derived by decrypting the transaction data field using hash codes of the smart contract function signatures.

Mühlberger et al. [46] adopt a similar approach, also utilizing Caterpillar for identifying process instances and activities on Ethereum. Their primary focus, however, is on generating log files. The procedure starts by gathering and hashing the signatures of all process instance smart contract functions. Then, they collect blocks from an Ethereum client via Remote Procedure Calls (RPC) [2], filtering the transactions in these blocks based on the hashed signatures. Transactions are organized by process instance, corresponding to the smart contract address from the worklist factory. The transaction hash values are decoded to extract data, which is then structured into event and trace attributes. This structured data is exported as an XES log [13], with the option to include additional information like block timestamps in the XES log generation.

Koschmider et al. [47] propose a process mining method for Hyperledger blockchain, independent of existing blockchain-based BPMSs. In their approach, each block is viewed as a process instance and its transactions as potential activities. The process involves retrieving blocks as JSON objects, extracting read/write operations by iterating through blocks, and identifying state changes to document corresponding events with predefined attributes. These events are then converted into CSV files for process model discovery using Disco [48] and conformance checking of smart contracts with ProM [49].

Corradini et al. [50], focusing on conformance checking, scan the Ethereum blockchain for auditable contracts, selecting those with significant transaction activity and user interaction. They collect transaction lists in JSON format [3] and consider the process instance as the aggregation of all transactions linked to a user and the smart contract. Transactions are grouped by sender and timestamp to form user-specific traces, which are formatted in XES and analyzed using three process mining algorithms (Heuristics miner, Inductive miner, and Split miner) with tools like Apromore and ProM. The resulting models represent various scenarios for

[2]https://www.rfc-editor.org/info/rfc5531
[3]https://www.rfc-editor.org/info/rfc7159

subsequent conformance checking.

Klinkmüller et al. [19] introduce a framework with modules for extracting blockchain logs and generating smart contract events, based on a manifest file detailing event data logging in the blockchain. The framework's validator module checks the manifest's correctness, and transaction logs are filtered to obtain specific smart contract data, which is then structured into event data and exported in XES format. The framework also includes cost-effective log generation techniques and was tested on Ethereum's CryptoKitties Dapp[4].

Expanding on this, Klinkmüller et al. [51] develop a configurable logging framework with enhanced extraction capabilities, not limited to process data. Similar to its predecessor, it uses a manifest for extraction and log generation but additionally includes transaction-triggered data. The framework supports fine-grained queries through various filters and offers different output formats (XES, CSV, TXT). It was tested in case studies like Ethereum network statistics, monitoring Augur Dapp, and analyzing Cryptokitties Dapp for process visualization and conformance.

Müller et al. [52] focus on process mining for decentralized applications, emphasizing control flow and organizational aspects on Ethereum. They use the Solidity events API and transaction data to gather event data, defining process instances as the collection of smart contract events in a transaction log. The approach categorizes activities based on involved actors and was tested to provide insights into Ethereum's usage and complexity.

Wirawan et al. [21] aim to incorporate transaction time in blockchain-generated event logs to account for temporal process properties. Their framework comprises several workflows: Extraction, Decoding, Process Mining, and Transition System Analysis. The Extraction workflow aggregates transactions, which are then decoded and converted to XES format. The Process Mining workflow generates process models evaluated for trace fitness, and the Transition System Analysis workflow enhances the model with time data, analyzing process states and time metrics.

The analysis of existing works revealed varying methods for the different blockchain logging steps followed by the family the works belong to. Key observations included:
Pre-blockchain

- **Step 1** (Data Collection): Varied according to the data source, with BPMSs the tool are already available, while other systems required custom scripts to aggregate event data.
- **Step 2** (Logs Creation): This was either straightforward in systems with

[4]https://www.cryptokitties.co/

existing BPMS (Business Process Management Systems) that contain a log creation functionality, or in the case of other systems the creation of logs requires custom scripts.

- **Step 3** (Ingestion of logs into blockchain): Logs are typically uploaded to a blockchain by sending them to a smart contract, using either a communication middleware or a custom application. One method involves an event-focused blockchain, where logs are ingested as transactions with each block representing an event.

- **Step 4** (Querying blockchain for logs): This is done either by call smart contracts functions, using a client specific to each blockchain platform or by extending the BPMS to include this functionality.

- **Step 5** (Application of Process Mining): In the works examined, there was no retrieval of the ingested data for testing, and as a result, no process mining techniques were utilized.

Post-blockchain

- **Step 1** (Data Collection): Varied according to the blockchain used, with Ethereum-based approaches benefiting from an active community and existing tools, while Hyperledger approaches required custom scripts.

- **Step 2** (Process Instance Identification): This was either straightforward in blockchain-based existing BPMSs, where smart contracts are identified as the process instance, or required manual decision-making and assignment in other cases.

- **Step 3** (Activities Identification): Ranged from using business process smart contract functions in BPMSs to decoding data in smart contract events and using predefined rules for combining various blockchain data into activities.

- **Step 4** (Logs Creation): Most approaches structured the gathered event data in XES or CSV log files.

- **Step 5** (Application of Process Mining): The approach which used the gathered event logs primarily focused on process discovery and, in some cases, conformance checking.

Tables 3.2 and 3.3 summarise the comparison of the works of each family according to their approaches for each step.

Table 3.2: Comparison of the pre-blockchain family

	Step 1		Step 2		Step 3				Step 4		Step 5	
Used method	Tools	Custom script	Tools	Custom script	Dedicated application	Middleware	Smart contract	Transaction submission	Dedicated extension	Smart contract call	Discovery	Conformance checking
[40]	x		x			x	x		x			
[39]		x		x	x		x			x		
[41]	x		x				x			x		
[42]	x		x					x				
[43]	x		x				x					

Synthesis Our work aligns with previous studies that employed predefined rules to map blockchain data into event logs. Nevertheless, our research uniquely concentrates on artifact-centric data, a focus not covered in earlier mentioned studies. To do so, we establish specific mapping rules designed to construct artifact-centric elements, including artifacts and their relations, from blockchain data. Furthermore, we automate the generation of logs structured according to our novel logging format. In our work, we also focus on smart contracts broadly as an event data source, rather than limiting our examination to those specifically within BBPMs. This choice is adopted due to the larger volume of smart contracts present on blockchain platforms that are not part of BBPMS and the fact that they present a more complex challenge in reconstructing event data. This complexity arises because the code of these smart contracts typically does not include mechanisms designed to produce structured event logs.

3.4 On discovering artifact-centric process models

In this section, we investigate the works related to the discovery of artifact-centric process models. We also include the works that focused on object-centric processes.

Berti and van der Aalst [53] introduce Multiple Viewpoint Models (MVPs) for representing process interconnections, transcending traditional case notion constraints. MVPs integrate various case notions into a unified model, annotated with

Table 3.3: Comparison of the post-blockchain family

	Step 1		Step 2		Step 3				Step 4		Step 5	
Used method	Tools	Script	Smart contract address	Manually assigned	Transactions	Predefined list	Smart contract events	object changes	XES	CSV	Discovery	Conformance
[50]		x		x	x					x		x
[47]	x			x		x		x		x	x	x
[46]	x		x		x				x		x	x
[44]	x		x		x							
[19]	x			x			x		x		x	x
[51]	x			x	x		x		x	x	x	x
[52]	x			x		x	x				x	
[21]	x			x	x		x		x		x	x

frequency and performance data, offering a comprehensive view of process dynamics. To utilize process mining effectively, activity-centric event logs are derived from these MVPs. MVP construction involves synthesizing Event-to-Object (E2O), Event-to-Event (E2E), and Activity-to-Activity (A2A) graphs, with E2O linking events to objects, E2E connecting events, and A2A associating activities. A key feature of MVPs, especially in the A2A graph, is their ability to link activities and label these connections with object classes and frequencies. However, each object class's graph is created independently, leading to separate graphs for each class. While these graphs are presented together, they don't explicitly show interactions among different object classes.

Nooijen et al. [54] present a methodology for extracting artifact life-cycle models from event data in data-centric systems, particularly focusing on ERP systems. Their approach is structured into several phases, starting with the extraction of event data from databases, which includes event specifics, case identifiers, and their interrelations. This initial phase is essential for defining the analysis scope. The next phase involves identifying key process data objects (artifacts) and associated events, critical for understanding entity interactions within the system. Subsequently, the data is divided into XES event logs for each artifact, enabling detailed analysis of each artifact's life-cycle. The final phase applies conventional process discovery techniques to these logs to build individual process models for each data object.

However, this method does not address the interactions between different data objects, a limitation for fully understanding the overall process dynamics.

Fahland [55] use event knowledge graphs to model behavior over multiple entities. They argue that entities are a more general term then object and they use it because they work invloves the study of the behavior of entities that are not tangible objects. Their discovery approach involves a graph-based approach where, instead of creating entire traces linked to a single case identifier, local directly-follows relations are established for each entities. Events can be part of multiple such relations, depending on their correlations with different entities. This results in event knowledge graphs where paths of directly-follows edges can intersect, unlike in classical event logs where traces are disjoint. Additionally, they use querying and aggregation on the event knowledge graphs to get insights into the behaviors. They also use querying to obtain different representations from object-centric directly-follows graphs to Proclets. The data source they use is event tables and they project to adapt their approach to databases.

Lu et al. [56] focus the problem of process mining in complex Enterprise Resource Planning (ERP) systems, where processes involve multiple interrelated business objects, each with its own identifier and behavior. To address this, the paper presents a semi-automatic approach for analyzing ERP system data. This method involves identifying artifact-centric process models that describe the life-cycles and interactions of various business objects within ERP systems. The approach includes steps for discovering individual artifacts from relational data, extracting event logs for each artifact, and identifying interactions between these artifacts to form a comprehensive process model. The discovery is done by using the flexible Heuristics Miner on XES logs.

The work of van Eck et al. [57], introduces the Composite State Machine Miner (CSM Miner), a tool developed to shift the focus of process discovery from the traditional activity-centric view to a state-centric view, particularly for processes with multiple perspectives. The CSM Miner aims to discover and analyze state-based models, where each state represents a combination of states across different process perspectives. The tool, implemented as a plug-in for the ProM framework, inputs XES event logs where each event signifies a state change. It constructs both a composite state machine representing the overall process and individual state machines for each perspective. The tool allows interactive exploration of these models, providing statistics and insights into state occurrences, transitions, and interdependencies between perspectives.

The approach of Popova and Dumas [58] discovers unbounded synchronization

conditions in artifact-centric process models, specifically GSM models, from event logs. Synchronization conditions appear in artifact-centric processes where an artifact's state change depends on the states of a varying number of other artifacts. The proposed approach focuses on inter-artifact synchronization, where the guard condition of one artifact's stage may depend on the milestones of other artifacts. However, they do not discover other data conditions nor consider reflexive interactions. The approach uses artifact synchronization logs to capture interactions between artifacts and applies a decision tree algorithm to extract synchronization rules.

Nguyen et al. [59] propose an approach to for business process stage identification and process model discovery from event logs. Their stage identification approach essentially groups activities into stages through graph cuts. It transforms the log into a Directly-Follows Graph (DFG), then partitions this graph into stages. The aim is to maximize modularity, ensuring high internal connectivity within stages and minimal connectivity between different stages. The process discovery approach involves mining individual submodels for each stage and then sequentially chaining them to form a cohesive process model. However they do not discover stages' nesting and rely only on directly follows relations.

Popova et al. [23] propose an approach for converting Petri Net models into GSM models. It uses existing algorithms for mining Petri Nets to discover the life cycles of individual artifacts, which they then represent as GSM models. The core of the method involves extracting the immediate ordering relations between transitions in a PN, translating them into conditions, and incorporating them into sentries assigned to the guards of GSM stages. However, they do not take into account the interactions between different artifacts and consider the data conditions of the petri-net as provided. They also do not discover the hierarchy between stages, i.e., they only consider atomic stages. The different levels of abstraction of operations are thus not discovered.

In Table 3.4, each approach is evaluated based on the specific characteristics of artifact-centric process mining, providing insights into their strengths and limitations. The first column references the works we are comparing, the second column indicates the type of event data used as input by each approach, and the third column shows the process model representation used to display the output of the discovery. The last column indicates whether the discovered process model presents artifacts interactions.

Synthesis The majority of related work tends to overlook the explicit representation of interactions between artifacts, if they consider such interactions at all. In contrast, our work not only uncovers interactions between artifacts but also distinguishes

Work	Data Source	Model Representation	Interactions
[53]	Activity-centric logs	MVPs	Partially
[54]	ERP system data	Individual artifact models	No
[55]	Event tables	Event knowledge graphs	Yes
[56]	ERP system data	Artifact-centric models	Yes
[57]	XES logs	State-based models	Yes
[58]	Artifact synchronization logs	GSM models	Partially
[59]	Event logs	Directly-Follows Graphs	No
[23]	Petri Nets	GSM models	No

Table 3.4: Comparison of Artifact-Centric Process Mining Approaches

multiple types of these interactions.

Additionally, while artifact-centric process models are utilized as a process representation in only two of the related works, their methodology pivots on first generating Petri Nets and then converting them into artifact-centric process models. Our method, on the other hand, directly extracts artifact-centric process models from event logs. Moreover, the novelty of our approach resides in its ability to discover not only the data conditions but also the hierarchical structuring of activities, aspects largely neglected in other studies.

When it comes to data sources, the prevalent reliance is on relational databases. Our approach stands out as the only one to leverage blockchain technology as a data source for artifact-centric process mining. Hence, it marks a significant departure from conventional data sources, with the aim to unlock the potential of blockchain for artifact-centric process mining.

3.5 Conclusion

We provided in this chapter an exploration of the existing approaches relevant to our work. We ordered the approach according to three themes: (i) On storing artifact-centric event data, (ii) On extracting event data from blockchain, (iii) On discovering artifact-centric process models. For each theme, we briefly presented each work and analyzed its contributions and compare it with ours. For the first theme, we presented the existing object-centric logging formats and we established that they present limitations when it comes to storing artifact-centric event data. These limitations include the lack of support for concepts like relations of objects evolution, as well as lack of optimization of log structure. We show for the second theme that none of the works consider the artifact-centric perspective. They focus solely on the activity-centric perspective which reveals a gap in literature related

to the extraction of artifact-centric event data from blockchain. The analysis of the works related to the third theme, indicates that no approach discover artifact-centric process models directly from event logs, i.e., without needing data external to event logs. Additionally, they do not discover data conditions nor nesting of stages. In following chapters, we will present our contributions to fill the research gaps we found in the literature. In Chapter 4, we propose a logging format which overcomes the limitations of the related object-centric logging format. Our approach to extract artifact-centric event data from blockchain data is presented in Chapter 5. In Chapter 6, we propose a novel approach to discover artifact-centric process models from artifact-centric event data without the need for domain knowledge.

Chapter 4

Artifact-centric event logs

Contents

4.1 Introduction

This chapter presents our approach for storing artifact-centric event data and thus answer $RQ1$ (How to capture efficiently artifact-centric event data ?). Toward this end, we propose an artifact-centric logging format which captures all information available in artifact-centric event data. This format was designed by extending the OCEL standard [17] to support the missing concepts of relations, and objects and relations evolution (see Section 3.2).

In this chapter we present the limitations of these formats and motivate our choice to use OCEL as a starting point to propose our new format (Section 4.2). Noticeably, the DOCEL [38] format, introduced in Section 3, presents similarities with our proposed format. This format was introduced in the literature after our novel format [35]. However, we will also use its limitations to present our novel format. Drawing from these limitations we present our new format called ACEL (Artifact-centric Event Log) which addresses them through new concepts (Section 4.3). We present a qualitative evaluation of our novel model to capture efficiently artifact-centric event data by comparing a sample ACEL log of Crytokitties to XOC and OCEL logs of the same application (Section 4.4). We also evaluate the potential of ACEL in solving process mining challenges in the context of artifact-centric processes (Section 4.5), before concluding (Section 4.6).

The work in this chapter was published in the IEEE SCC conference [35].

4.2 Limitations of current logging formats

In Section 1.3, the challenges arising from using XES to store object-centric and artifact-centric event data were highlighted. In this section we delve in detail into these challenges, and their impact, and show the limitations of the formats proposed to solve them.

To illustrate these limitations, we rely on the description of Cryptokitties in the whitepaper [60]. This description shows that the Breeding activity affects two kitties and the Birth actvities affects the same kitties that bred together in addition to a new born kitty. Hence the Breeding event is associated with two instances of the kitty artifact, one referred to as 'sire' and the other as 'matron', and the Birth event is associated with the same two instances along side a new instance of the same artifact, referred to as 'kitten'. The Breeding event affects the two kitties with the following changes: (i) the cooldownPeriod [1] of both kitties is updated with the

[1]The cooldownPeriod is the time a kitty needs to wait before breeding again.

same value, and (ii) Both kitties are linked through a relation that prohibits them to breed with other instances of the kitty artifact before the Birth event. The Birth event affects the two kitties with the following changes: (i) the cooldownPeriod of 'sire' and 'matron' is updated to zero, (ii) the relation between 'sire' and 'matron' is ended allowing them to breed again with other kitties, and (iii) the new born 'kitten' is affected two properties, its genes with a permanent value and its owner which can change. Tables (4.1, 4.2, 4.3), 4.4 and 4.5 depict the storage of the Breeding and Birth events in XES, XOC and OCEL, respectively. In particular, Tables 4.1, 4.2, 4.3 are the traces of one 'matron', one 'sire' and one 'kitten', respectively within a flattened XES log. The flattened XES logs was obtained by selecting the kitty artifact as the main case notion and then filtering the events to select those relevant to the kitty perspective. The kitty identifier was chosen to be the process instance identifier (trace), thus the events were grouped by the id of each kitty instance.

EventId	**Activity**	**Timestamp**	**sireId**	**matronId**	**kittyId**	**cooldown Period**	**genes**	**owner**
E1	Breeding	23/10/23 06:11:51	1475706			11225643		
E2	Breeding	23/10/23 06:11:51		1240424		11225643		
E3	Birth	24/10/23 10:12:36	1475706		1576916	0		
E4	Birth	24/10/23 10:12:36		1240424	1576916	0		
E5	Birth	24/10/23 10:12:36	1475706	1240424			8658320...	0xf12A13..

Table 4.1: Flattened XES log: trace of a 'matron'

EventId	**Activity**	**Timestamp**	**sireId**	**matronId**	**kittyId**	**cooldown Period**	**genes**	**owner**
E1	Breeding	23/10/23 06:11:51		1240424		11225643		
E2	Breeding	23/10/23 06:11:51	1475706			11225643		
E3	Birth	24/10/23 10:12:36		1240424	1576916	0		
E4	Birth	24/10/23 10:12:36	1475706		1576916	0		
E5	Birth	24/10/23 10:12:36	1475706	1240424			8658320...	0xf12A13..

Table 4.2: Flattened XES log: trace of a 'sire'

In the following sections, we explore the limitations of the logging formats, particularly focusing on the challenges that may arise when storing artifact-centric event data. To elucidate each point, we will refer to the tables presented earlier.

EventId	Activity	Timestamp	sireId	matronId	genes	owner
E1	Birth	24/10/23 10:12:36	1475706	1240424	8658320...	0xf12A13..

Table 4.3: Flattened XES log: trace of a 'kitten'

Event	Event Type	References	Object Model	
			Objects	Relations
E1	Breeding	o_1, o_2	$(id = o_1, class = cat, tokenId = 1240424,$ $genes = 6789232...,$ $cooldownPeriod = 11225643)$ $(id = o_2, class = cat, tokenId = 1475706,$ $genes = 7507913...,$ $cooldownPeriod = 11225643)$	$(r1, o_1, o_2)$
E2	Birth	o_1, o_2, o_3	$(id = o_1, class = cat, tokenId = 1240424,$ $genes = 6789232..., cooldownPeriod = 0),$ $(id = o_2, class = cat, tokenId = 1475706,$ $genes = 7507913..., cooldownPeriod = 0)$ $(id = o_3, class = cat, tokenId = 1576916,$ $genes = 8658320..., cooldownPeriod = 0)$	$(r2, o_1, o_3)$ $(r3, o_2, o_3)$

Table 4.4: Tabular representation of artifact-centric event data in XOC

4.2.1 Deficiency

Deficiency is the disappearance of events, initially present in the event data, from the event log. This issue only happens when using XES to store artifact-centric or object-centric event data, as the initial event data needs to be flattened. In Table 4.3, showing the trace of a kitten, the events related to breeding of its sire and matron are not available. One might argue that this is not a case of Deficiency as the events are all linked to the same artifact and if all traces are placed in the same log the Breeding events would be accounted for. However, we argue that this is indeed a case of deficiency but on the trace level, i.e., on an object instance level. Indeed, this deficiency at the object instance level would still cause process discovery issues, as the discovered lifecycle for each kitty would still be missing the causality between the Birth and the Breeding activities. This deficiency is present in this example due to the particularity of the example we chose, where interactions exist between instances of the same artifact. We note here that this type of processes were not considered in previous works, to the best of our knowledge.

The deficiency issue is avoided when using XOC, OCEL and DOCEL to store the same events. As shown in Tables 4.4, 4.5 and 4.6, all events are within the same log. This is due to the fact that both formats support multiple case notion and do not force the separation of events in traces.

Table 4.5: Tabular representation of artifact-centric event data in OCEL

Event Identifier	**Activity**	**Timestamp**	**Attribute**		**Objects**
			Name	**Value**	
e1	Breeding	23/10/23 06:11:51	Resource	0xf12A13..	o_3, o_4
e2	Birth	24/10/23 10:12:36	Resource	0xf12A13..	o_5, o_6

(a) Events

Object Identifier	**TokenId**	**Type**	**genes**	**cooldownPeriod**
o_1	1240424	kitty	6789232..	0
o_2	1475706	kitty	7507913..	0
o_3	1240424	kitty	6789232..	11225643
o_4	1475706	kitty	7507913..	11225643
o_5	1240424	kitty	6789232..	0
o_6	1576916	kitty	8658320..	0

(b) Objects

4.2.2 Convergence

Convergence happens when object events referring to multiple instances of the selected case notion are replicated. This replication can lead to misleading diagnostics because it appears as though more events occurred than actually did. This is observed in the XES sire and matron traces in Tables 4.1 and 4.2. In Table 4.1, two Breeding events and three Birth events are present. This is due to the fact that the Breeding event affects both the matron and the sire and the Birth event affects the matron, the sire and the kitten. Since Table 4.1 is the trace of the matron, the events Breeding and Birth are duplicated for the other instances.

Similarly to the previous issue, this one is avoided when using XOC, OCEL and DOCEL. Tables 4.4, 4.5 and 4.6 clearly show no duplication of events are within the same log. This is due to the fact that both formats support the association of multiple objects to one event.

4.2.3 Divergence

Divergence arises when events linked to distinct instances of an object, that was not chosen as the case notion, are still viewed as being causally connected due to their association with an instance of the object selected as case notion. Essentially, this means that events linked to various instances of one object are combined in the log, making it challenging to clearly understand the causal links between individual events and the specific objects they involve. This is not apparent in the example we have because of the condition stating that two kitties that bred together can not

Table 4.6: Tabular representation of artifact-centric event data in DOCEL

Event	**Activity**	**Timestamp**	**kitty**
e1	Breeding	23/10/23 06:11:51	{1240424, 1475706}
e2	Birth	24/10/23 10:12:36	{1240424, 1475706}

(a) Events

Kitty

kittyId	**genes**
1240424	6789232..
1475706	7507913..

(b) Objects and static attributes

CooldownPeriod

CooldownPeriodId	**CooldownPeriod**	**EventId**	**kittyId**
$cl1$	0	e1	1240424
$cl2$	0	e1	1475706
$cl3$	11225643	e2	1240424
$cl4$	11225643	e2	1475706

(c) Dynamic attributes

breed again with other kitties until the Birth event happens. If this rule was not in place, we could have seen in the XES log several Birth events involving several matrons being duplicated in the trace of the sire. This is due to the inability of the XES format to accommodate multiple case notions and the absence of support for events being associated with multiple objects. Thus, this would have been avoided by using XOC, OCEL or DOCEL.

4.2.4 Denormalization

Denormalization is the flattening of data structures when converting an object-centric log in a classic format like XES. This leads to duplicated data without any possibility for referencing as the data from multiple related objects is combined into attributes of an event. Referencing allows a relation to be indicated by foreign keys instead of duplicating all the attributes of the target object of a relation into the the source object of this relation. It also allows to store all the constant attributes of an object in the log and only reference the id of the object. In Tables 4.1, 4.2 and 4.3, it is clear that all the attributes of the three instances, sire, matron and kitten, are duplicated and grouped throughout the events. In particular in the Birth event in Table 4.3, it is impossible to know if the attribute genes belong to the sire or the kitten nor that the sire is linked to the kitten. This issue results from the absence of support for objects and relations by XES. Consequently, when used in

an artifact-centric context, these flattened logs fail to reveal artifact lifecycles and interactions among them.

XOC, OCEL and DOCEL partially solve this problem. Both of them use object structure to group attributes of one entity. However, OCEL and DOCEL do not support relation as shown in Table 4.5 and 4.6, and and XOC duplicates the relational model with each event.

4.2.5 XOC or OCEL or DOCEL or a new format ?

The issues outlined above pose substantial obstacles to achieving a smooth implementation of object-centric or artifact-centric process mining. These issues become notably pronounced during the discovery phase as established in Section 1.3. The challenges arising during the discovery phase, specifically pertaining to artifact-centric discovery technique, are examined in Chapter 6.

The shortcomings during the discovery phase have cascading effects on subsequent phases of the BPM lifecycle, such as the redesign phase. An erroneous discovered model might lead to enhancements in the wrong process, while a correct model, in term of activity ordering, lacks a comprehensive data perspective necessary for process analysis. One might suggest that a solution to the last problem would be to enrich the activity-centric model with data objects to gain a data perspective. However, these attempts might not align with the actual data perspective of processes, posing paradigm challenges. The disconnection between activity-centric and artifact-centric representations is analogous to the challenge faced by a developer transitioning between programming paradigms. Indeed, when designing or reworking these processes, designers prioritize understanding how artifacts behave, i.e., how they evolve or interact, rather than determining the sequence or parallelism of activities. This means that the traditional activity-centric approach doesn't align well with designing artifact-centric processes, posing challenges for designers. To illustrate, switching between these representations is akin to the challenges faced by a developer transitioning from the object-oriented programming paradigm to the functional paradigm. The shift requires adapting from a familiar structure to a new approach, causing difficulties for those accustomed to the former. The mitigation of information loss regarding objects and relation, and the avoidance of convergence and divergence in event data, requires the use of an artifact-centric event log or an object-centric event log format like OCEL, DOCEL or XOC. These formats focus on storing information about objects and their relationships to events, thereby linking each event to a list of associated objects. However, these formats also exhibit limitations in the context of artifact-centric processes, which we explore below.

OCEL lacks the capability to capture the lifecycle of artifacts and fails to represent object evolution through events. Additionally, it overlooks storing information about relationships between artifacts, making the analysis of artifact interactions challenging. The format also struggles to capture activity triggering through transitions, potentially misrepresenting the relations between activities and objects in resulting logs. Triggering through transitions occurs when an activity causes the execution of another activity because the objects they each affect are associated through a relation. For example, if an activity A_1 impacts an object O_1 and subsequently affects another object O_2 due to its relation with O_1, by resulting in a subsequent action A_2 influencing O_2, we can say that A_1 triggered A_2 through transition. In an OCEL log, this scenario might be represented in two ways: firstly, where A_1 is linked to both O_1 and O_2, and A_2 is associated solely with O_2; or secondly, where A_1 is connected to O_1, and A_2 is linked to O_2. The discovery of an artifact-centric or object-centric model from these logs would give the following : In the case of the first log the model will represent $A1$ as part of the lifecycles of both objects and this is not conform to reality; In the case of the second log, the resulting model will present no link between both activities through the objects and this interaction would be lost.

Similarly to OCEL, DOCEL does not support relations between artifacts and thus does not allow the discovery of artifact interactions. There is also no support for activity triggering in this format. Additionally, DOCEL logs, as illustrated in Table 4.6 present redundancy in the storage of data like the duplication of the events and attributes identifiers in the dynamic attributes table. Indeed, the storage of the values of each dynamic attribute in a separate table is not an efficient structuring of the log and it hurdles scalability. Furthermore, there is no implementation of this format and thus no an algorithm to generate event logs in this format. The lack of a specification for this formats and extraction tools renders applicability impossible.

On the other hand, XOC supports relations, allows for the possibility to record object changes by storing their attributes with each event, and implicitly acknowledging transitions when events refer to related objects. However, it does not store the state of objects and thus only partially support object evolution. Additionally, the format suffers from scalability issues due to storing the complete object model within each event element. This redundancy impedes scalability, especially in an artifact-centric process context where processes are driven by artifact changes. Furthermore, because XOC has been implemented only in XML, it lacks adequate support for structured data elements such as lists and maps essential for effective data object structuring.

Although XOC and DOCEL support more concepts related to artifact-centric event data than OCEL, their design of the format is not efficient and not scalable. On the other hand, OCEL support data structures like lists and maps and decouples the object elements from the events. For this reason we choose to build on OCEL, benefiting from its more efficient design, to obtain a more optimal logging format than the existing ones where there is no redundancy.

4.3 Artifact-centric event logs

In the previous section, we discussed our decision to create a new format based on extending OCEL. This extension, called Artifact-Centric Event Log (ACEL), aims to capture all the important aspects of artifact-centric event data. ACEL is designed to fill in the gaps in OCEL, particularly in dealing with object evolution and object relations. By adding concepts like **lifecycle**, **Object changes**, **Relations**, and **Relation Changes** to OCEL, we aim to address ***RQ1***. These concepts are represented in Fig. 3.1 where new concepts are depicted in red and existing OCEL concepts are in white. To illustrate the new concepts, we use Tables 4.7a, 4.7b, and 4.7c which depict the storage in ACEL of the same Breeding and Birth events we used for the other formats.

In the following, we provide a detailed description of these newly introduced concepts.

4.3.1 Object Change

Object changes are the modifications that happen to artifacts' attributes when they are affected by events. For instance, in the GSM model of the motivating example (see Section 2.1.2.1), after the Breeding and Birth events the attribute 'cooldownPeriod' of both 'sire' and 'matron' changes value. This alteration in the properties of artifacts, as mentioned in Section 1.1, is how we differentiate between artifact-centric processes and object-centric processes. Thus, we consider that artifacts are objects which are subject to changes after the occurrence of events. Since, the changes happen on the attribute level of the objects, a logging format that aims to store artifact-centric event data need to store the value of object attributes after each event.

This is supported in XOC, as shown in Table 4.4 where the attributes' values are stored with each new event, even if the old values are also stored with each new event (redundancy). However, this not the case with OCEL, where object changes are not supported. Therefore, if we were to store this information in OCEL, we would

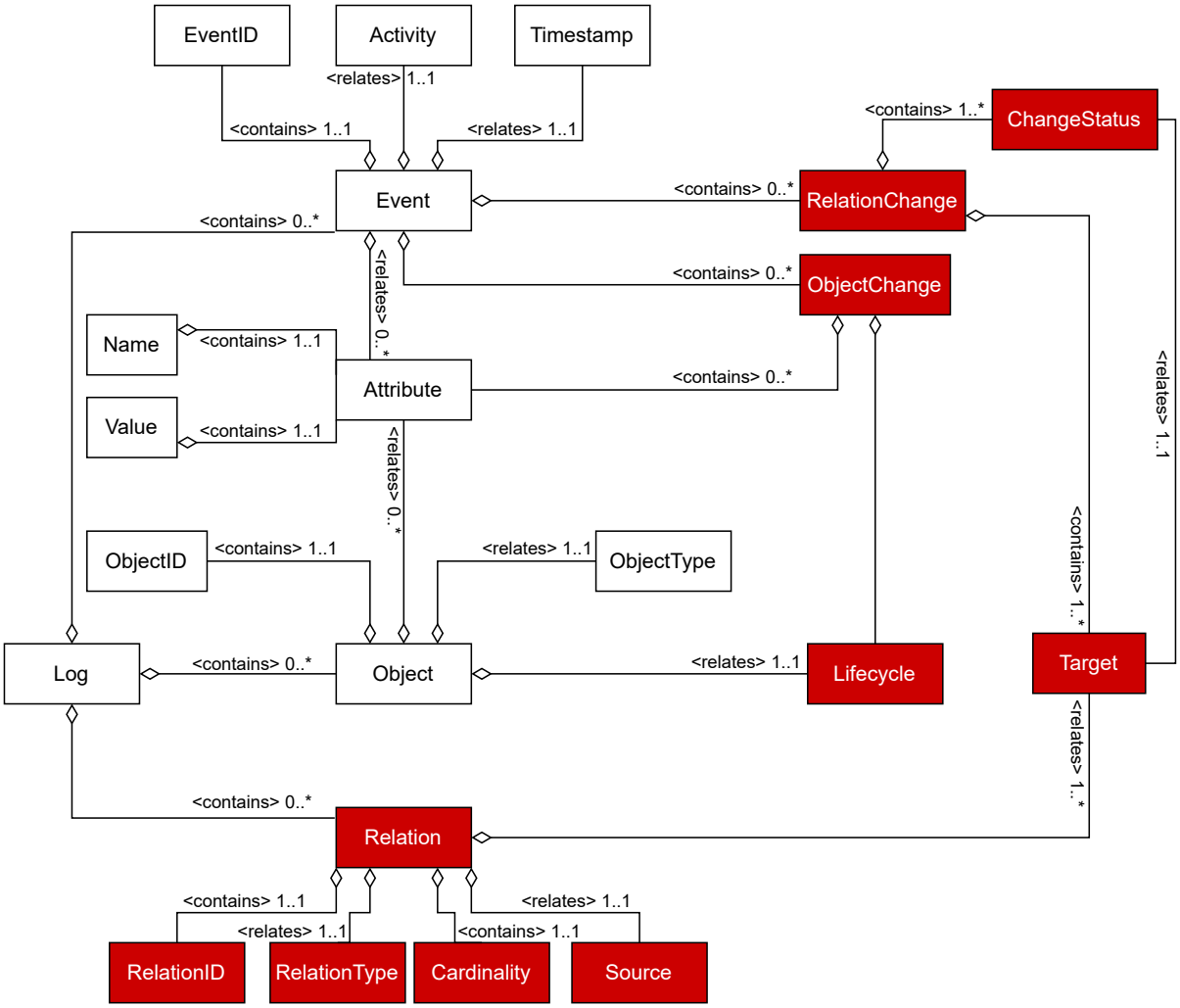

Figure 4.1: ACEL model (based on OCEL model)

need to create new objects. In other words, a new object with a new identifier is generated each time there is a modification in the objects' attributes. For instance, in Table 4.5, when updating the cooldownPeriod attribute of the kitty object $o1$, a new object $o3$ is created, even when this evolution does not necessarily lead to the creation of new objects. This results in information loss regarding object evolution and leads to redundant entries by rewriting unchanged attributes with the same values.

To address this in ACEL, we distinguish between two attribute types: *static* and *dynamic*. Static attributes, such as the genes of a kitty, remain unchanged once the object is created and are stored in object elements. However, dynamic attributes can change post-object creation, through events. With ACEL we use dynamic attributes to track object changes, i.e., object evolution, avoiding the unnecessary creation of multiple objects for each change.

In ACEL, we store static attributes in object elements, while we store dynamic attributes in a list named *Object changes* within event elements. As illustrated in Tables 4.7a and 4.7b, this method ensures that only modified attributes are recorded in the Object Changes list, minimizing redundancy, that is encountered in XOC, and preserving the evolution of objects. For instance, contrary to Table 4.5b, ACEL creates only two objects representing distinct kitties with static attributes ("Type" and "genes") and only the altered attribute values are stored in the Object Changes list.

4.3.2 Lifecycle

In artifact-centric processes, each step in an artifact's lifecycle leads it to a new state, which holds a specific semantic meaning within the process. The recording of object states is another particularity of artifact-centric processes over object-centric processes, as they represent the semantic meaning of object changes. These states are necessary to achieve defined business goals by defining goal states, i.e., final states, to prevent unwanted behaviors. This is the case in certain processes, where one artifact can reach a particular goal state only after another artifact has reached another specific goal state [61]. For instance, in the motivation example, after the Breeding event, the 'sire' can only be in the state of father after the 'matron' it bred with gives birth (Birth event) and becomes in the state of 'Mother'. This particular attribute is not explicitly supported by XOC, as it is not specific to artifact-centric processes. Table 4.4 shows that although the changes of the objects are recorded, no state is recorded for them. OCEL also lacks the support of this concept as it is missing the concept of object evolution, which is strongly linked to the notion of

state.

To address this gap with ACEL, we introduce a new essential dynamic object attribute called *lifecycle* for object evolution tracking. It records the state of an artifact following an event occurrence. We store the lifecycle attribute within Object Changes list as it is a dynamic attribute. Table 4.7 shows how we stored the state of the sire after the Birth event in the Object Changes list as an attribute 'lifecycle' with the value 'BecameFather'.

4.3.3 Relation

As mentioned in Section 1.1, object and artifact differ in the role they play in their processes. Artifacts refer to a tangible or conceptual entity that undergoes various states and transformations throughout a business process. Artifacts are the focal points around which processes are structured. Consequently, they differ from objects in the nature of the relations linking them throughout the processes. While artifact-centric processes focus on the lifecycle and state transitions of key business entities, object-centric processes offer a more holistic view, capturing the intricate web of interactions among all entities involved in a process. The relationships in artifact-centric processes are defined by the state transitions and interactions of these artifacts. Each artifact has a lifecycle that describes its progression through different states, driven by business rules or process activities. Artifacts often interact with each other, influencing each other's state transitions. These interactions can be sequential, parallel, or conditional, and they define how artifacts co-evolve throughout the process. The relationships between objects in an object-centric process are more complex and multidimensional compared to artifact-centric processes. Objects can be associated with multiple events, and an event can relate to multiple objects. This many-to-many relationship adds a layer of complexity to how objects interact and influence each other within a process. The relations are not just defined by state transitions but also by the interconnected nature of objects participating in various events and activities.

XOC provides the possibility to store relations as part of events but OCEL does not define relations for objects. However, relations establish important connections between objects following an event. For instance, in the motivating example, after the Breeding event, a link is formed between the kitty and the other kitty it bred with. These connections carry vital information that can impose constraints on how an artifact's lifecycle progresses, such as prohibiting the other kitty from engaging in further breeding activities before giving birth. In ACEL, relations are defined for an object using a mandatory static attribute named *source*. Additionally, relations

encompass a required static attribute called *cardinality*, which can assume values such as *One2Many* or *One2One*. In XOC, the cardinality is not made mandatory as shown in Table 4.4 where it is missing.

4.3.4 Relation Changes

Relations link one or multiple objects to a *source* through a mandatory dynamic attribute we call *target*. This dynamic attribute, *target*, is recorded in the relation changes list (see Tables 4.7a and 4.7c) as as relations can evolve due to events, through additions or deletions of one or more targets. To capture these changes, we introduce a mandatory attribute called *changeStatus* for relation changes. This attribute, associated with the target attribute, specifies the nature of the change ('addedTarget' or 'deletedTarget'). This concept enables us to track the evolution of an artifact's relations, documenting changes over time. Notably, a single relation's change list can accommodate multiple target and changeStatus pairs, allowing for the addition of new targets and deletion of previous ones. This evolution of relations is not supported by XOC as shown in Table 4.4, where only the source and target of the relation are referenced.

In summary, ACEL comprises events, objects, and relations, illustrated in Fig. 4.1. Each one of these elements contains an identifier and attributes. Object and Relation elements store attributes capturing details about their creation. Similarly to OCEL, every event element contains an activity, a timestamp, and optional additional attributes. An event can impact one or multiple objects and/or relations, referenced by their identifiers in the event's list of objects and/or relations, potentially leading to changes. These changes are logged alongside the event within the ObjectChanges and/or RelationChanges lists, as shown in Table 4.7a.

4.4 Qualitative Evaluation of the ACEL model

The aim of our proposed logging format is to fully capture artifact-centric event data for the purpose of process mining. To effectively discover a model of a specific process type, it is important to use a logging format that is rich enough to capture all the required information (the correct logging format). To answer requirements of process mining for event data, the logging format should structure event data accurately, i.e., according to reality, and not introduce convergence and divergence. Also, to ensure storage efficiency, it is important that the logging format does not contain any redundant information. In short, the logging format should be **rich,**

Table 4.7: Tabular representation of artifact-centric event data in ACEL

EventId	Activity	Timestamp	Attribute		Objects	Relations
			Name	Value		
e1	Breeding	23/10/23 06:11:51	Resource	0xf12A13..	1240424, 1475706	r1
e2	Birth	24/10/23 10:12:36	Resource	0xf12A13..	1240424, 1475706 , 1576916	r2, r3

ObjectChanges			RelationChanges		
ObjectId	Attribute	NewValue	RelationId	Target	ChangeStatus
1240424	lifecycle	Pregnant	r1	1475706	addedTarget
1240424	CooldownPeriod	11225643			
1475706	lifecycle	FutureFather			
1475706	CooldownPeriod	11225643			
1240424	lifecycle	BecameMother	r2	1240424	addedTarget
1240424	CooldownPeriod	0	r3	1475706	addedTarget
1475706	lifecycle	BecameFather			
1475706	CooldownPeriod	0			
1576916	lifecycle	Born			
1576916	Owner	0xf12A13..			

(a) Events

ObjectId	Type	genes
1240424	kitty	6789232..
1475706	kitty	7507913..
1576916	kitty	8658320..

(b) Objects

RelationId	Type	Source	Cardinality
r1	siringWith	1240424	One2One
r2	hasMother	1576916	One2One
r3	hasFather	1576916	One2One

(c) Relations

accurate and concise. In this section, we evaluate qualitatively the richness, accuracy and conciseness of ACEL. As baselines, we consider the XES, XOC, and OCEL standards. To compare the logging formats we rely on the listings 8.1, 8.2, 8.3 and 8.4 which show respectively XES, XOC, OCEL, and ACEL logs we manually created according to the structure of each logging format. Based on the results of this evaluation, we also discuss the potential of our logging to improve the results of artifact-centric process mining.

Listing 4.1: Excerpt of an ACEL Log

```
1 {"acel:global-event":{"acel:activity":"__INVALID__"}...
2 "acel:events":{"1":{"acel:activity":"Breeding",
3 "acel:timestamp":1511415679, "acel:vmap":{"resource":
4 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
5 "acel:omap":["1240424"],
6 "acel:rmap":["r1","r2","r3"],"acel:ocmap":{"1240424":
7 {"CooldownPeriod":"11225643","lifecycle":"Pregnant"},
8 "1475706":{"CooldownPeriod":"11225643",
9 "lifecycle":"FutureFather"}},
```

```
10 "acel:rcmap":{"r1":{"target":"0",
11 "changeStatus":"deletedTarget"}...}},
12 "2":{"acel:activity":"Birth",
13 "acel:timestamp":1511415679, "acel:vmap":{"resource":
14 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
15 "acel:omap":["1576916"],"acel:rmap":["r1","r2","r3"],
16 "acel:ocmap":{"1576916":{"owner":
17 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936",
18 "lifecycle":"Born"} ...},
19 "acel:objects":{"1240424":{"acel:type":"kitty",
20 "acel:ovmap":{"genesSequence":
21 "626837621154801616088980922659877168609154386318304496692374110716999053"}}...},
23 "acel:relations":{"r1":{"acel:type":"siringWith",
24 "acel:rvmap": {"source":"1240424",
25 "cardinality":"oney2one"}}....}}
```

The **richness** of ACEl is first apparent compared to XES and OCEL, through the presence of relations in the ACEL log excerpt in Listing 8.4 (lines 7 and 25), while the others cannot store that information. It is then seen between ACEL and XES in terms of object count because XES does not support the concept of object contrarily to ACEL as shown in Listing 8.4 (lines 6 and 21). It is also visible between ACEL and OCEL in the possibility to count traces in the ACEL log and not in the OCEL one because OCEL does not support the notion of object evolution but rather store attribute changes by the creation of new objects as shown in Listing 8.3 (lines 6 and 16). In XOC this information appears but is redundant and no clear indication of state is made, while in ACEL the attribute 'lifecycle' provides that information (see lines 8 and 10 of Listing 8.4).

Listing 4.2: OCEL Log Snippet

```
1 {"ocel:global-event":{"ocel:activity":"__INVALID__"}...
2 "ocel:events": {"1":{"ocel:activity":"Breeding as Matron",
3 "ocel:timestamp":1511415679,
4 "ocel:vmap":{"resource":
5 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
6 "ocel:omap":["o1"]},
7 "2":{"ocel:activity":"Breeding as Sire",
8 "ocel:timestamp":1511415679,
9 "ocel:vmap":{"resource":
```

```
10 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
11 "ocel:omap":["o2"]},
12 "3":{"ocel:activity":"Birth as Matron",
13 "ocel:timestamp":1511415679,
14 "ocel:vmap":{"resource":
15 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
16 "ocel:omap":[o3]}...
17 }},
18 "ocel:objects":{"o1":{"ocel:type":"kitty","ocel:ovmap":
19 {"genesSequence":
20 "626837621154801616088980922659877168609154386318304496692374110716999053"}}...}}
```

The **conciseness** of ACEL is clear when comparing it to XOC, as ACEL does not duplicate unchanged values contrarily to XOC (see lines 16 and 39 of Listing 8.2). It is also clear in both the event and object count when compared to XES and OCEL, respectively(see lines of and lines of). First, the object count in the ACEL log is inferior to that of the OCEL one because contrary to OCEL, ACEL does not create a new object each time there is a change. Second, as compared to XES, ACEL stores fewer events. This is due to the fact that in order to indicate the presence of relations between objects, we need to define new events for each change occurring to each object, even if it does not correspond to the execution of an activity in the process.

Listing 4.3: XOC Log Snippet

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <log xoc.version="1.0">...
3 <event>
4         <string key="id" value="e1"/>
5         <string key="activity" value="Breeding"/>...
6         <model><objects>
7         <object><string key="id" value="o1"/>...
8         </object>
9         <object>
10                <string key="id" value="o2"/>...
11        </object></objects>
12        <relations><relation>
13                <string key="id" value="e1-r1-o1"/>...
14        </relation></relations>
```

```
15          </model>
16          <references>
17                  <object><string key="id" value="o1"/>
18                  </object>
19                  <object><string key="id" value="o2"/>
20                  </object>
21          </references>
22  </event>
23  <event>
24          <string key="id" value="e2"/>...
25          <model><objects>
26          <object><string key="id" value="o1"/>...
27          </object>
28          <object><string key="id" value="o2"/>...
29          </object>
30      <object><string key="id" value="o3"/>...
31          </object></objects>
32          <relations>
33          <relation><string key="id" value="e1-r1-o1"/>...
34          </relation>
35      <relation><string key="id" value="e2-r2-o3"/>...
36          </relation>
37       ...
38          </relations></model>
39          <references>
40                  <object><string key="id" value="o1"/>
41                  </object>
42                  <object><string key="id" value="o2"/>
43                  </object>
44                  <object><string key="id" value="o3"/>
45                  </object>
46          </references>
47  </event>
48  </log>
```

This also testifies to the **accuracy** of the ACEL log in comparison with the XES log. If we consider the Birth activity of the process, we find that it corresponds to one event (Birth) in the ACEL log (see line 12 of Listing 8.4) and three events

(Give Birth as Matron, Give Birth as Sire, Is Born) in the XES log (see lines 20, 28 and 38 of Listing 8.1). The event Birth of a kitty in ACEL changes three kitties lifecycles to born, becameFather and becameMother, and introduces two relations hasFather and hasMother linking the born kitty to his parents. This testifies to a more accurate representation of the process, since it does not introduce new activities while capturing object evolution.

Listing 4.4: XES Log Snippet

```
1 <log xes.version="1.0" xes.features="nested-attributes"
2 openxes.version="1.0RC7">...
3 <trace><string key="ident:piid" value="1240424"/>
4 <event>
5     <int key="logIndex" value="1"/>
6     <string key="concept:name" value="Conceive as Matron"/>
7     <string key="Activity" value="Breeding"/>...
8     <string key="sireId" value="1475706"/>
9     <string key="cooldownPeriod" value="11225643"/>...
10 </event>
11 <event>
12     <int key="logIndex" value="2"/>
13     <string key="concept:name" value="Conceive as Sire"/>
14     <string key="Activity" value="Breeding"/>...
15     <string key="matronId" value="1240424"/>
16     <string key="cooldownPeriod" value="11225643"/>...
17 </event>
18 <event>
19     <int key="logIndex" value="3"/>
20     <string key="concept:name" value="Give Birth as Matron"/>
21     <string key="Activity" value="Birth"/>...
22     <string key="kittyId" value="1576916"/>
23     <string key="sireId" value="1475706"/>
24     <string key="cooldownPeriod" value="0"/>...
25 </event>
26 <event>
27     <int key="logIndex" value="4"/>
28     <string key="concept:name" value="Give Birth as Sire"/>
29     <string key="Activity" value="Birth"/>
30     <string key="Resource" value="0xf12A13.."/>
```

```
31      <string key="kittyId" value="1576916"/>
32      <string key="matronId" value="1240424"/>
33      <string key="cooldownPeriod" value="0"/>
34      <date key="time:timestamp" value="2023-10-24T10:12:36"/>
35  </event>
36  <event>
37      <int key="logIndex" value="5"/>
38      <string key="concept:name" value="Is Born"/>
39      <string key="Activity" value="Birth"/>...
40      <string key="matronId" value="1240424"/>
41      <string key="sireId" value="1475706"/>
42      <string key="genes" value="8658320..."/>
43      <string key="owner" value="0xf12A13.."/>...
44  </event>
45  </trace></log>
```

It is important to note that all the information present in XES, XOC and OCEL is also present in ACEL and with a simple processing we can obtain XES, XOC or OCEL logs from an ACEL log. ACEL logs on the other hand are richer since they contain information regarding the evolution of objects and their relations.

This qualitative evaluation shows clearly that ACEL achieves the objective of fully capturing artifact-centric event data.

4.5 ACEL for process mining: a qualitative evaluation

This provides a first confirmation of ACEL's adequacy for process mining based on the previous presentation of ACEL concepts (Section 4.3) and the example log (Listing 8.4). It shows the potential of ACEL to solve problems of process mining input data as follows:

4.5.1 Additional knowledge

The current event data formats do not store all the required information for artifact-centric process mining. For example in XES the relational model is not existing and for OCEL the relations between the objects is missing. Therefore, domain knowledge needs to be provided before artifact-centric process mining can be applied, especially

for the discovery of interactions between artifacts. As Listing 8.4 shows all this data is already present in ACEL logs.

4.5.2 Depiction of reality

Artifact-centric event data is not accurately represented by other logging formats. Although OCEL captures the multiple case notion, it does not capture the evolution of objects. XOC store the attribute changes of objects but omit to store the state of objects. Whereas in ACEL, as Listing 8.4 (lines 6-9) shows, object evolution is captured through tracking of attribute and state changes of each object.

4.5.3 Convergence

This problem is solved by OCEL but not in the context of artifact-centric event data when events of related artifacts objects are duplicated in an artifact-centric process instance [8]. In the example when we consider the case notion as kitty and we add the events of its related parents the Breeding activity **then** a breeding event for the cases associated with both parents will be duplicated for the kitty case. Moreover, this activity although it affects the lifecycle of kitty, is not part of it, so duplicating it is not realistic. In ACEL this is solved by tracking attribute changes. Therefore, if the state of a related object affects the lifecycle of the main object, the attributes can be accessed without duplicating the event.

4.5.4 Denormalization

If a log does not support the notion of objects nor object relations, the logs will present a form of relational denormalization, i.e., data is duplicated with no referencing possible. For example, because the artifact kitty has a father and a mother, the kitty's data will be written twice alongside the father's data and the mother's data. This is solved in ACEL because the relational model and explicitly linking attributes and their changes to their objects is supported.

4.5.5 Transition

Activities can be triggered through transitions, i.e., an activity A_1 affects an object O_1 and because O_1 has a relation with O_2 this leads to an activity A_2 affecting O_2. In our example, the Success of Auction activity affects the object kitty but is not part of both lifecycle. When an Auction is successful, the state of the kitty is

not changed but a transfer activity is triggered, which is an example of interaction between both artifacts.

4.6 Conclusion

In this chapter, we answered the research question $RQ1$, which is: How to capture efficiently artifact-centric event data?. We analysed the limitations of existing object-centric logging formats (XES, OCEL and XOC) when used to store artifact-centric data, and concluded that a new format is required. We used OCEL as a base for our new format because of the data structures and data exchange format it uses. To support the notion of object evolution without ambiguity, we extended OCEL with the possibility to track **object changes** and in particular using the **lifecycle** attribute which indicates the state of an object. We also added the concept of **relations** between objects to capture the interactions between lifecycles. We aslo introduced the concept of **dynamic and static attributes** to keep the objects elements separate from events, contrarily to XOC, by storing the attributes whose values are constant in the object elements. We used dynamic attributes, whose values can change with each event, to specify the nature of the change occurring to the objects and their relations while avoiding redundancy. We also quantitatively evaluated (a quantitative evaluation is provided in the next chapter) the potential of our new format through a qualitative comparison between ACEL and XES, XOC, and OCEL. In contrast to these formats we established that ACEL achieves a richer, more concise and accurate storage of artifact-centric event data. The results of our evaluation also showed that ACEL holds great potential to achieve more accurate artifact-centric process mining results. One limitation of this evaluation is that is was performed on only one use case. In next chapter will use two uses cases to show the potential of ACEL for process mining.

Chapter 5

Extracting artifact-centric event data from blockchain applications

Contents

5.1 Introduction

This chapter presents our approach for extracting artifact-centric event data from blockchain applications and generating logs in ACEL, our newly introduced format (cf. chapter 4), from the extracted data. This approach aims to answer the research question $RQ2$ (How to collect artifact-centric event data from blockchain applications ?). For that purpose, we rely on domain knowledge to ensure that the collected data is conform to the business reality. Our approach covers all the steps to transform raw blockchain data into ACEL event logs: the collection and decoding of raw data, and the mapping of this data into ACEL elements before storing in an ACEL log. Additionally, our approach is designed to support applications comprising many smart contracts, i.e., their process execution involves calling functions from different smart contracts. To fit this particularity, our approach includes an ordering step, according to the timestamp of the logged smart contract events. Indeed, querying blockchain platforms, such as Ethereum, for smart contracts events yield a batch result where event are grouped according to smart contracts instead of timestamp. Thus, we need to order them according to the timestamp to obtain an accurate ordering of events.

The extraction, as in similar works [19, 21, 51] presented in Section 3.3 is based on predefined rules for mapping smart contract event data to business process event data. We propose an extraction algorithm for the Ethereum blockchain as it is public and provides several ready to use smart contract logs. However, our algorithm can easily be adapted to other Blockchain platforms which support smart contract events.

In this Chapter, we also seek to study the perspective of ACEL for process mining, in particular for process discovery. For that purpose, we propose an approach to filter ACEL logs in order to adapted them to existing process discovery techniques which support OCEL logs as input (Section 5.4).

We evaluate our contributions, regarding the extraction of ACEL logs from blockchain and the adapting of the extracted logs for process discovery techniques, using two popular Ethereum applications: Cryptokitties [1] and Augur [2] (Section 5.5).

The work in this chapter was published in the IEEE SCC conference [35] and in the peer-reviewed elsevier SIMPAT journal [37].

[1] `https://www.cryptokitties.co/`
[2] `https://augur.net/`

5.2 Preliminaries

5.2.1 Ethereum

Ethereum [62] is a blockchain platform that was the first to support development of smart contracts, that run on the Ethereum Virtual Machine (EVM), and is widely used. EVM serves as the computational backbone that enables the interpretation and execution of smart contracts and transactions. It acts as a universal runtime environment for decentralized applications, ensuring that code execution is consistent and secure across the entire network. Ethereum supports the development of decentralized applications (DApps) and use a native cryptocurrency, called Ether (ETH). In Ethereum, there are two types of accounts: externally owned and contract accounts, each identified by a unique address. This blockchain platform is considered as a turing complete system, i.e., a computation model capable of solving any algorithmic problem provided there are enough resources like time and memory. However, while Ethereum is Turing complete in theory, real-world constraints such as gas limits (to prevent infinite loops) and the cost of executing code may place practical limitations on the complexity of computations that can be performed within a smart contract. In Ethereum, "gas" is a unit of measure for the computational work required to execute transactions and run smart contracts on the network. Complex operations, like storage writes or complex computations, consume more gas than simple operations. If a transaction runs out of gas before completing its execution, it's considered invalid, and any changes made by the transaction are rolled back.

Smart contracts in the Ethereum context are programs written in a stack-based bytecode language, but other high- level languages can also be used, with Solidity [3] [4], being the most commonly used. The code of the high-level languages is compiled to bytecode that is executable on the EVM. They are deployed on the blockchain and self execute in order to automatically facilitate, verify, or enforce the terms of a contract or agreement between parties not trusting of each other without the need for intermediaries. Additionally to the code, each smart contract has an account, a balance and a private storage. The storage and balance constitute the state of the smart contract which is modified when the contract is invoked [63] through transaction sent to his address. The function of a smart contract can be called by sending a transaction to the contract's address with the function name and the input data. To know the name of the functions and their required input data, Ethereum

[3] Solidity is an object-oriented smart contract programming language influenced by C++, Python and Javascript

[4] `https://docs.soliditylang.org/en/v0.8.23/`

clients rely on the smart contract's ABI (Application Binary Interface) [64]. The ABI defines how to interact with a smart contract, specifying the methods, their inputs, and outputs in a binary format. It enables communication and interaction with Ethereum smart contracts from external applications and languages. When invoked, the smart contract's functions can generate events, which are recorded in the blockchain as logs.

Transactions in the Ethereum network, are the fundamental actions that involve the transfer of Ether or the execution of smart contracts. Each transaction contains crucial data such as sender and receiver addresses, the amount of Ether transferred, gas fees, the timestamp, and a unique transaction hash. It also possesses a data/-payload field, which is empty for transactions related to Ether transfer. However, when interacting with a smart contract, this field contains encoded data specifying the function to be executed and any parameters associated with that function.

5.2.2 Event data in Ethereum

The primary source for business process event data in Ethereum is transaction data and smart contract event data. Fig 5.1 shows how this data is strucutured within the blockchain.

Since **Ethereum transactions** [62] contain detailed information, including timestamps, transaction IDs, involved parties, and actions taken. This information allows for precise tracking of user interactions with an application, making it ideal for reconstructing processes and identifying bottlenecks or inefficiencies. Furthermore, real time tracking of this extracted transaction data provide insights into specific business-related events and allows companies to monitor their process execution to ensure compliance with their reference model. There are two types of transactions related to smart contracts : regular and internal. Regular transactions result from an external account (user) call to a smart contract function. The transactions are entirely recorded in the blockchain's blocks. Internal transactions on the other hand are the result of a smart contract account call to another smart contract's function. When a smart contract function incorporates a call to another smart contract function, it is executed and any generated events in the latter contract are produced and recorded as part of the blockchain logs. However, it is important to note that there exists no explicit trace of this function call within the transaction data itself. Thus, internal transactions are notably absent from the blockchain's blocks. This differentiation is purposefully designed within publicly available Ethereum client implementations. Accessing this information would necessitate the modification of the client's behavior, i.e., rewrite the code of the client to keep a trace of internal

transactions.

Smart contract events [33] serve the purpose of conveying detailed information regarding the execution of functions within the EVM and their associated outcomes. Additionally, they have the capacity to relay information pertaining to events that occurred during the execution process to external off-chain components [5]. The raw data [6] of these smart contract events is hashed as it underwent a hashing process before being stored on the blockchain. The data of each event include a topic, i.e the event title (or name) and its ordered parameters, and the associated data, as exemplified in Fig 5.1. It is noteworthy that each event is unique. This uniqueness implies that while two events may share identical names and parameter sets, they can differ in the arrangement of these parameters.

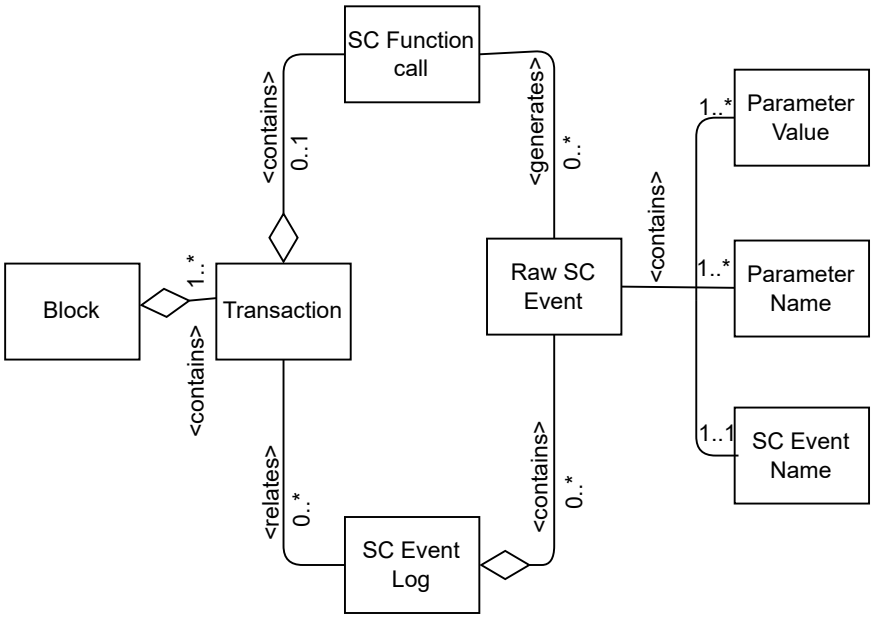

Figure 5.1: Informal representation of Ethereum event data

The definition and configuration of these events is done by the smart contract developers, who possess the flexibility to define the events' ordering and the number of parameters, within the constraints set forth by the supported types of the smart contract language. Smart contract events allow the storage of information about all types of function calls, including those of internal transactions. Indeed, smart contract events resulting from internal transactions are preserved for reference.

[5]Services or systems outside of the blockchain network.

[6]Raw blockchain data is the unprocessed data that is directly stored on the blockchain

5.3 Extracting ACEL logs from Ethereum

In the context of Ethereum, unlike transactions, particularly in the case of internal transactions, smart contract events are always available and thus they provide a more complete picture of the execution of a process. Furthermore, smart contract events are used to emit information about the result of a smart contract function's execution and therefore they may contain more data. For this reason we chose smart contract events as our source to collect the event data that we will rely on to apply process mining. In the following we detail our approach to collect artifact-centric event data from Ethereum and render it adequate for process mining techniques through the generation of ACEL logs from this data. We divide our approach into three phases, as illustrated in Fig 5.2: we start by a configuration phase (Section 5.3.1) to gather domain knowledge from experts, then we proceed to obtain event data from blockchain in the collection phase (Section 5.3.2), finally we map the collected event data into ACEL elements from which we generate ACEL logs in the mapping & generation phase (Section section 5.3.3).

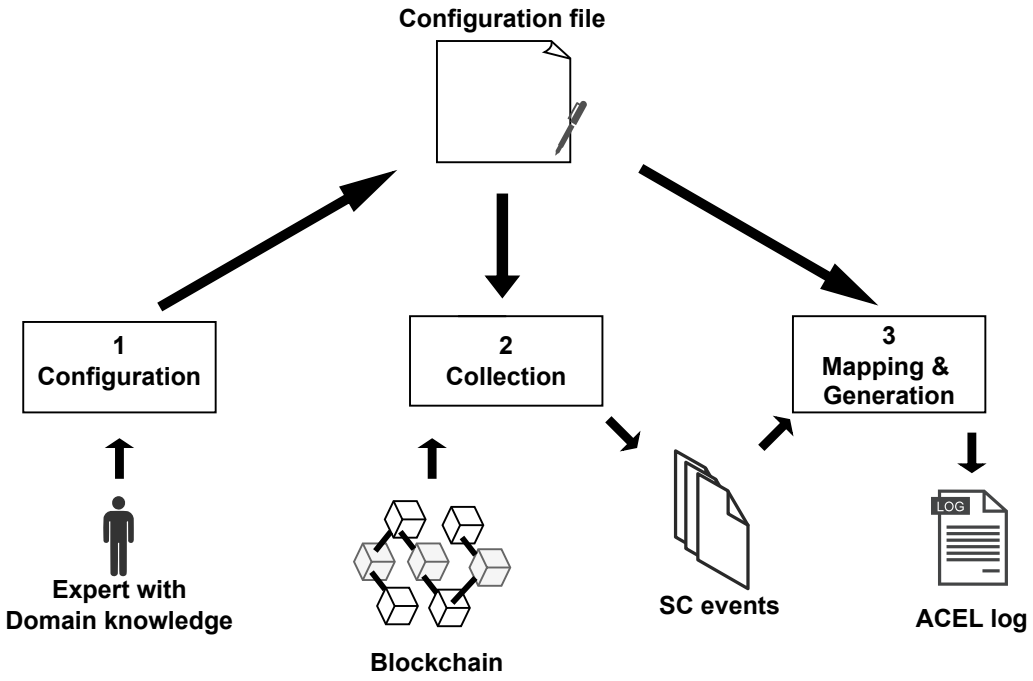

Figure 5.2: Overview of the approach to extract ACEL logs from Ethereum

5.3.1 Configuration phase

As mentioned in Section 5.2, unlike transactions, particularly in the case of internal transactions, smart contract events are always available and thus provide a more

complete picture of the execution of a process. Indeed, we cannot access smart contract function calls present in internal transactions but the smart contract events emitted during that call can be retrieved. Furthermore, smart contract events are used to emit information about the result of a smart contract function execution and therefore they contain more data. For this reason we choose smart contract events as our source for event data. The collection and the mapping & generation phases require some domain knowledge about the code of the application's smart contracts to collect event data and the relational model underneath to reconstruct artifacts and their relations. The role of the configuration phase is to gather the data required by the collection and the mapping generation phases. The collection phase requires data about the smart contracts of the application. In general, a blockchain application can comprise many smart contracts calling each other to execute a transaction, thus we need the *address* of each relevant smart contract. The ABI is also required to interact with the smart contract and retrieve its event logs. ABIs are not stored on the blockchain and only the developer can provide that information. Some Blockchain explorers [7] like Etherscan [8], an explorer of Ethereum, are used to publish this information but it is not always the case, thus we prefer to rely on user information. We provide flexibility in the choice of data to be collected, i.e., the user can choose which batch of data to extract, though a configuration parameter named Block Range. The Block Range allows the user to specify the block from which the extraction starts and the block at which it ends.

The mapping process involves the conversion of the collected smart contract events into ACEL elements (Artifacts, Relations, Events), which requires a precise understanding of which parameters of the smart contract events align with the attributes within each ACEL element. Therefore, a reference information model is imperative to delineate the structure of artifacts and their relations. The most accurate way to obtain this information is to rely on domain experts, which will specify the types of Artifacts, their attributes and the relations linking them.

We group all the required domain knowledge in a configuration file that serves, as illustrated in Fig 5.2, as input to the collection and the mapping & generation phases. To guarantee that users who follow our approach will provide the right information, i.e., required domain knowledge structured according to the specificities of our approach, we propose a structured template to be followed to reduce the manual effort and errors when creating a configuration file.

[7]Blockchain explorers are web-based tools that provide a user-friendly interface for accessing and inspecting the details of individual blocks, transactions, and addresses on a blockchain network, allowing users to track the status of transactions and analyze the blockchain's activity.

[8]`https://etherscan.io/`

The structure of our proposed template is shown in Fig 5.3. It shows that a configuration file contains one or many SC elements (smart contract elements), this is due to the fact that a blockchain application can run through multiple smart contracts. The elements contained in the configuration file element are:

- A SC element contains information about the smart contract, i.e., its *address* and *ABI*, and the *blockrange*. A *blockrange* is composed of two variables (*startBlock*) and (*andBlock*). SC elements also contain Artifact, Relation and SC Event elements.
  - An Artifact element contains information about an artifact, i.e., its type (object type as per OCEL nomenclature) and its attributes.
  - A Relation element contains information about a relation between artifacts, i.e., its name (object type), the source and target artifacts and its cardinality, e.g., one to one or many to one.
  - A SC Event element contains a SC Event Topic which provides information about a smart contract event. It also contains an Event Mapping from the smart contract event information to ACEL elements.
    * A SC Event Topic contains the name of the smart contract event and the ordered list of its parameters' names and their types as they are written in the smart contract code.
    * An Event Mapping contains three mappings: an ACEL Event mapping, an ACEL Object mapping, and an ACEL Relation mapping.
    * An ACEL Event mapping contains information about the activity whose execution generated the event: its name, the timestamp and potential attributes. The name which is static and provided by the user, the timestamp is by default mapped from the block timestamp, and the mapping of the smart contract event parameters to the event attributes is defined by the user as it requires domain knowledge.
    * An ACEL Object mapping contains information about each object modified by the event. It contains the type of object, to be chosen from the list of Artifacts previously provided, the lifecycle of the object which is a static value provided by the user, and a mapping from the smart contract event parameters to the object attributes defined by the user.
    * An ACEL Relation mapping contains information about the created or updated relations between objects after an event occurrence. It

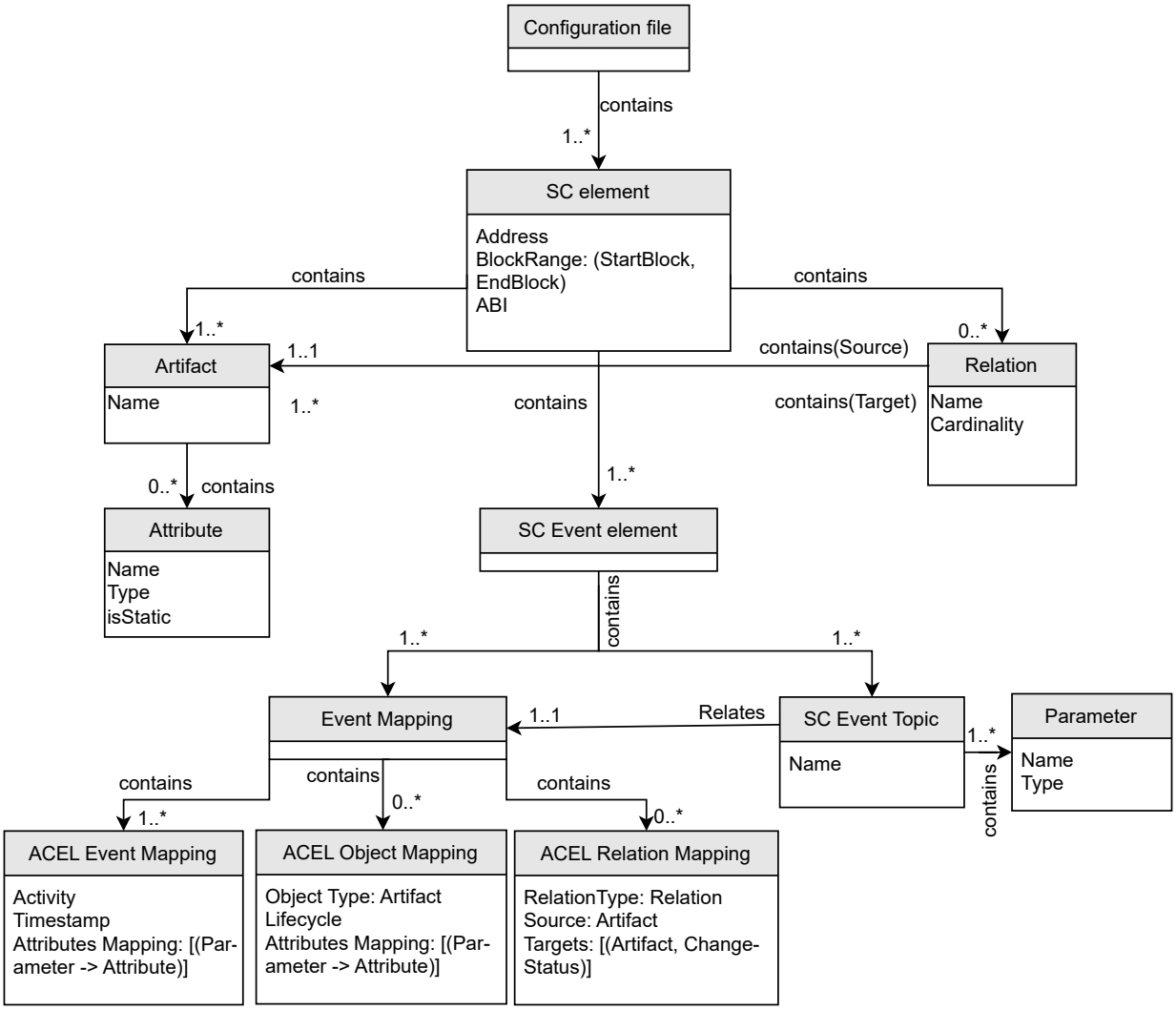

Figure 5.3: Structure of the configuration template

contains information about the relation type, to be chosen from the list of Relations previously provided, the type of the source artifact, and the list of targets. For each target, the type is chosen from the Artifacts pool and the Change Status is provided by the user, as it requires domain knowledge.

5.3.2 Collection phase

The collection phase consists of two steps: the connection to a blockchain network and the querying of the latter for smart contract events. We automate this process as described in Algorithm 1.

There are two ways to connect to an Ethereum network, either through a local node or a remote one. To start a local node, we need to run an Ethereum client on

our own machine and inevitably join the network. Remote nodes on the other hand are clients running on the cloud and made available by service providers. Most of these service providers impose a daily limit on the number of calls to a node depending on the purchase plan. The advantage of using a service provider over the use of a local node is the elimination of the configuration and maintenance efforts as well as the need for significant storage and network resources. Based on the previous argument we chose to design our approach for a scenario where remote nodes are used.

The connection to the network through a remote node requires the use of Remote Procedure Calls (RPCs)[9] and a connection key for a blockchain client (node) provided by the service provider. Thus, the API of the service provider and a connection key are the input of Algorithm 1, along with the $SC_{elements}$ taken from the configuration file. In lines 4-5 of Algorithm 1, we establish a connection to a remote node using the connection key and and RPC call through the service provider's API.

Once the connection is established we move to the second step of our collection phase, that is the querying of Ethereum. As mentioned in Section 5.3.1, this step requires the user provided data during the configuration phase. Specifically, we need the information related to the the smart contracts in order to fetch the batch of smart contract events available within the block range. We limit the required information to the addresses and ABI of the smart contracts and a block range (see SC element in Fig 5.3). In lines 6-7 of Algorithm 1 we use $SC_{elements}$ the list of smart contract elements (output of configuration phase) to query the blockchain via the node connection established in the previous step. The blockchain query, executed via the function $node.getAllEventsInRange()$ for each smart contract of the application, consists of fetching $List_{rawSC_{events}}$ all the raw smart contracts events, within the block range, of each smart contract using its $address$, ABI, $startBlock$ and $endBlock$. As explained in Section 5.2, the raw smart contract events are hashed. Thus, we use in lines 8-9 $Decode()$ a decoding function to reverse hash every smart contract event ($rawscev$) using the ABI. The result is $scev$ a human readable smart contract event that we add to $List_{SC_{events}}$ the list of decoded smart contract events in line 10. Function E is used in line 11 to map each $sc_{element}$ to its $List_{SC_{events}}$. After the mapping, we proceed to sort the list of events by their timestamp (line 12) using the function $sortByTimestamp()$.

[9]https://www.rfc-editor.org/info/rfc5531

Algorithm 1: Collection of Smart Contract Events

Input : $SC_{elements}$ a set of n smart contract elements, such that $\forall sc \in SC_{elements}, sc = (address, ABI, startBlock, endBlock)$;
$serviceProviderAPI$ the API of the service provider ;
$connectionKey$ a connection key for a blockchain node made available by the service provider

Output: E a function that maps elements of $SC_{elements}$ to subsets of SC_{events}, where SC_{events} is a set of m smart contract events

$E \leftarrow \{\}$;
$List_{SC_{events}} \leftarrow \{\}$ ▷ The list of decoded smart contract events ;
$List_{rawSC_{events}} \leftarrow \{\}$ ▷ The list of hashed smart contract events;
$node \leftarrow$ serviceProviderAPI.connect($connectionKey$) ▷ Connection to a remote blockchain node;
If $notEmpty(node)$
  ForEach $sc \in SC_{elements}$
    $List_{rawSC_{events}} \leftarrow$ $node$.getAllEventsInRange($address, ABI, startBlock, endBlock$);
    ForEach $rawscev \in List_{rawSC_{events}}$
      $scev \leftarrow$ Decode($rawscev, ABI$) $List_{SC_{events}} \leftarrow List_{SC_{events}} \cup scev$ ▷ The decode function uses the ABI to reverse hash raw smart contract events ;
    $E(SC_{elements}) \leftarrow List_{SC_{events}}$;
    $E \leftarrow sortByTimestamp(E)$ ▷ This sorting function orders event by timestamp ;
return E;

5.3.3 Mapping & generation phase

Similarly to the collection phase, the subsequent mapping & generation phase requires domain knowledge. All knowledge pertinent to this process, e.g, artifacts, relations, ACEL elements mappings, is encapsulated within the configuration file, as illustrated in Fig 5.3. To automate this phase we propose Algorithm 2 which takes as input $ConfigFile$, the configuration file (output of configuration phase), and $List_{SC_{events}}$, the list of decoded smart contract events (output of collection phase), and gives as output Log, an ACEL log.

Algorithm 2: Event Data Extraction and Generation of ACEL logs

Input: $ConfigFile$, $List_{SC_{events}}$

Output: $Log = (E, O, R)$

▷ ACEL log composed of event, object and relation elements **foreach** *sc* ***in*** $configFile.SC_{elements}$ **do**

    ▷ Constructing events ;

foreach *scev* ***in*** $List_{SC_{events}}$ **do**

  $ev \leftarrow$ **getMatchingEventMapping**$(scev, sc.SC_{events}Elements)$;

  if *notEmpty*(ev) **then**

    $e \leftarrow$ **initializeEvent**();

    $e.EA \leftarrow$ **getEventAttributes**$(scev, ev.AcelEventMapping)$;

    $e.O \leftarrow$ **getObjectList**$(scev, ev.AcelObjectMapping, ConfigFile.artifacts)$;

    $e.R \leftarrow$ **getRelationList**$(scev, ev.AcelRelationMapping, ConfigFile.relations)$;

      ▷ Constructing objects ;

    foreach *ob* ***in*** $e.O$ **do**

      $attribs \leftarrow$ **getObjectAttributes**$(scev, ev.AcelObjectMapping)$;

      foreach *att* ***in*** *attribs* **do**

        if *isStatic*$(att, ConfigFile.artifacts)$ **then**

          $o \leftarrow$ **getOrInitializeObject**$(scev, ev.AcelObjectMapping, ob)$;

          $o.OA \leftarrow$ **getObjectStaticAttribute**$(scev, att)$;

          $Log.os$.**add**(o);

        else

          $oc \leftarrow$ **getObjectDynamicAttribute**$(scev, att)$;

          $e.OC$.**ob**.**add**(oc);

        $lc \leftarrow$ **getObjectLifecycle**$(scev, ev.AcelObjectMapping, ob)$;

        $e.OC$.**ob**.**add**(lc);

      ▷ Constructing relations ;

    `// repeat lines 10-19 for each` *or* `in` $e.R$`...;`

      ▷ Generating log ;

    $Log.es$.**add**(e);

return *Log* ;

Our algorithm uses smart contract events to construct event, object and relation elements and store them in an ACEL log. We start with a filtering phase where we iterate over the smart contract events given as input and retain only those that have been listed in the configuration file given as input (lines 4-6). Then, we proceed to construct the event elements by affecting each one of them an identifier (line 7), its attributes (line 8) and the identifiers to its associated objects (line 10) and relations (line 11). We extract the event attributes and the identifiers of objects and relations from each smart contract event using the mapping rules defined in the configuration file. The following step consists of constructing new object elements by affecting each one of them an identifier or retrieving old objects elements to update them and extracting their attributes from the smart contract events (lines 12-23). We place the extracted static object attributes in the object elements (line 17) and the extracted dynamic object attributes in an event list of changes identified by the object identifier (lines 20-21). The lifecycle (state) of each object is also placed in that list of changes (lines 22-23). In the subsequent step, we construct relation elements using the same procedure used for object elements (line 24). Finally, we complete the log generation by adding each constructed event element the events to the ACEL log (line 26). We add the object and relation elements to the log right after placing the static attributes (line 18) because that is the only information they contain, while the event element is placed last as it requires the dynamic attributes of all its associated objects and relations. In the last step, we return the generated ACEL log (line 27). A more detailed description of the Algorithm is available in Chapter 8.

5.3.4 Towards a Blockchain agnostic approach

To adapt our approach to blockchain platforms other than Ethereum, one has only to adjust the part of the algorithms that are specific to each platform. In the following, we provide guidelines for the adaptation to be made.

Connection to the blockchain platform (*Collection phase*) The choice of node type (local or remote) might depend on the options available for each platform, thus we suggest to evaluate the cost of each option. Furthermore, the connection request via RPC calls should be adapted to the specificity of each platform.

Blockchain query (*Collection phase*) Most blockchain clients provide the possibility to customize the queries. In Ethereum, the query option of getting all event logs at once is available. However, this might not be the case for other

platforms and thus the syntax and type of query should be adapted to each platform.

Smart contract events (*Configuration phase*) The configuration template we defined follows the Ethereum structure for smart contract events. Hence, the configuration template need to be modified to take into account a specific smart contract event structure for a certain platform.

Extraction Algorithm (*Mapping & Generation phase*) Naturally any modification in the input of the extraction and generation algorithm necessitates to adapt the functions of the latter to the new structure of the input. For example, any function which extracts data from the smart contract event is tailored for the latter's structure and it needs to be adjusted when that structure changes.

5.4 Object-centric process mining on blockchain data

ACEL is not supported by existing process mining techniques and in particular discovery techniques which are the focus of this thesis. Before proposing a new process discovery technique to support ACEL, we first sought to adapt it to the existing techniques in order to test some of its characteristics against existing formats like OCEL and show the potential of ACEL. This adaptation is necessary because although ACEL follows OCEL' structure, it is richer with new elements, e.g., lists of object changes and relation elements, that are not supported by the discovery techniques tailored for OCEL. Thus, we propose an approach to filter ACEL to make it compatible with these discovery techniques while maintain most of the artifact-centric information that make up its richness.

In this section we present our approach to adapt ACEL to existing process discovery techniques.

5.4.1 ACEL's adequacy for existing process discovery techniques

Current process discovery techniques can not take an ACEL log as input and are not designed to discover artifact-centric models from event logs [8]. However, we can still leverage the information about artifact-centric processes captured by ACEL logs through these techniques. Since ACEL is based on OCEL and the latter is supported

by some existing techniques[65], we opt for adapting ACEL to those techniques. We focus on one particular discovery algorithm [14], that discovers Object-Centric Directly-Follows multiGraphs (OC-DFG) [10]. The choice of this discovery algorithm is motivated by the fact that it was implemented in a tool which supports OCEL. Thus, in order to make ACEL compatible with this algorithm we need to filter it to obtain an OCEL log, i.e., remove the new concepts which alter the structure of OCEL logs and make it unrecognizable by existing techniques. However, this filtering would still allow us to leverage some artifact-centric process information captured by ACEL, namely the evolution of artifacts and their interactions at an activity level. In the following section, we will propose a filtering method to obtain OCEL log from ACEL while keeping a trace of object evolution.

5.4.2 Adapting ACEL to existing process discovery techniques

In our proposed filtering approach, the initial step involves the elimination of the relation concept. This entails the removal of relation elements from both the log and the event elements' lists of relations and relation changes. Subsequently, we eliminate concepts related to events that are unsupported by OCEL, specifically the lists concerning object changes, i.e., dynamic attributes. Notably, we preserve the object elements alongside their static attributes and the lists of objects in the events, as these aspects align with the OCEL standard. Consequently, we obtain a log structured in accordance with OCEL while retaining a trace of object evolution, even if this evolution is obscured in nature, due to the removal of dynamic attributes. Essentially, our filtering approach captures evolution at the activity level while omitting details at the attribute level. The resulting log enables the identification of events impacting an object's lifecycle but without the knowledge of the specific nature of these alterations. Thus, our filtering approach allows us to keep a trace of object evolution. Hence, we are able to evaluate the potential of ACEL to improve process discovery by using the OC-DFGs discovery algorithm [14].

5.5 Implementation and Evaluation

In this section, we present the implementation and evaluation of our approach to extract artifact-centric event data from blockchain applications and generate

[10] An OC-DFG displays layers of Directly-Follows Graphs (DFG) [66] with a DFG for each object type.

ACEL logs. We also evaluate our method to apply object-centric process mining on blockchain data by filtering ACEL logs.

5.5.1 Event data extraction and generation of ACEL logs: Proof of concept

We implement our extraction and generation approach as a node.js Web application available online (`https://lazy-jade-armadillo-cape.cyclic.app/upload/`) (see Fig 5.4) containing two modules: a configuration module and an extraction module. The configuration module is designed to help users create the configuration file, required as input for the approach, and to reduce manual efforts. The extraction module is meant to automate the collection and, the mapping & generation phases. For testing purposes, examples of configuration files (kittyConf.json and augurConf.json) and their corresponding ACEL log files (kittyACEL.jsonacel and augurACEL.jsonacel) are made available on the same Web application (`https://lazy-jade-armadillo-cape.cyclic.app/upload/#files`).

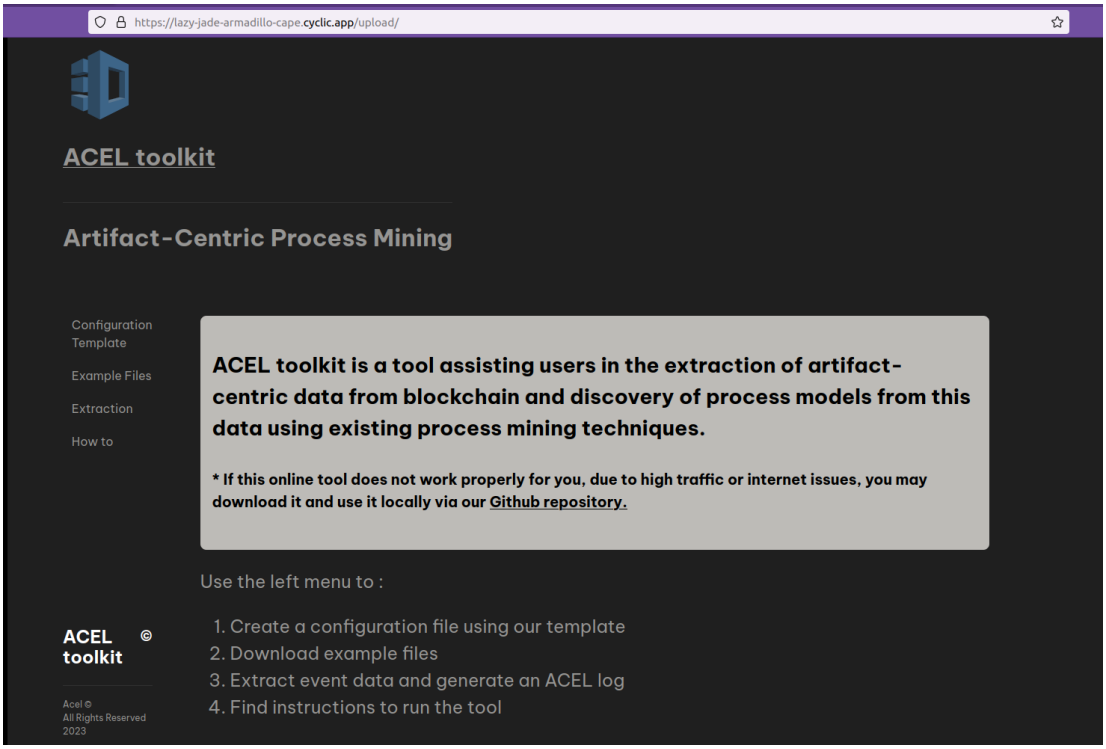

Figure 5.4: ACEL Tool

5.5.1.1 Overview of our tool

Configuration module This module serves as a user-friendly interface, guiding users through the creation of configuration files while performing validation checks to ensure compliance with our proposed template model (see Fig. 5.3). The resulting configuration file can be stored by the user and seamlessly employed as input for the extraction module within the same Web application.

To use the configuration file assistant, the user needs to click on the Configuration Template option from the left menu and then proceed to fill the form (see Fig 5.5). He will need to first provide a smart contract address, the ABI of the smart contract, and the start and end blocks. Secondly, he can create a new artifact by providing its name and the type of its identifier (Incremental or Extracted). Incremental identifiers require a base string (example for the artifact kitty, the base can be kit and the instances will be identified as kit1 kit2 kit3, etc), while extracted identifiers will be collected from the smart contract events. If the artifact has attributes they can be added by specifying their names, types and nature (static or dynamic).

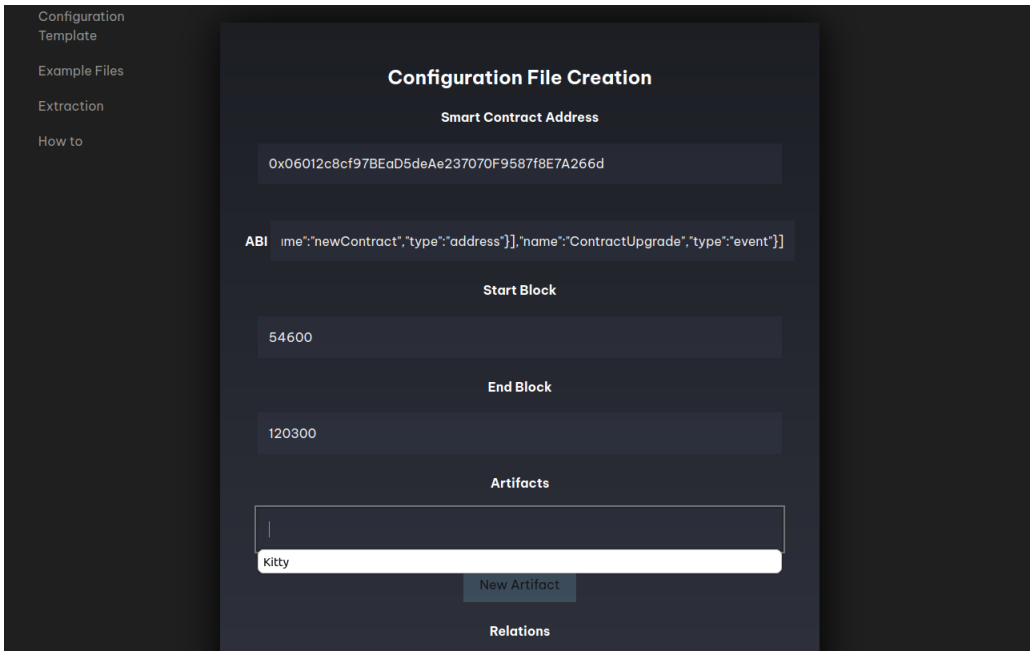

Figure 5.5: Configuration module: Smart contract element

Once a defined artifact is saved, its name will appear in the Artifacts list (see Fig 5.6), to be updated or used in the creation of relations. Similarly, relations are created by providing the base for their incremental identifiers, their source and target (selected from the list of artifacts), and their cardinality.

After the creation of artifacts and relations, the user can then create a new event and define mapping rules. He needs to provide the exact name of the smart contract

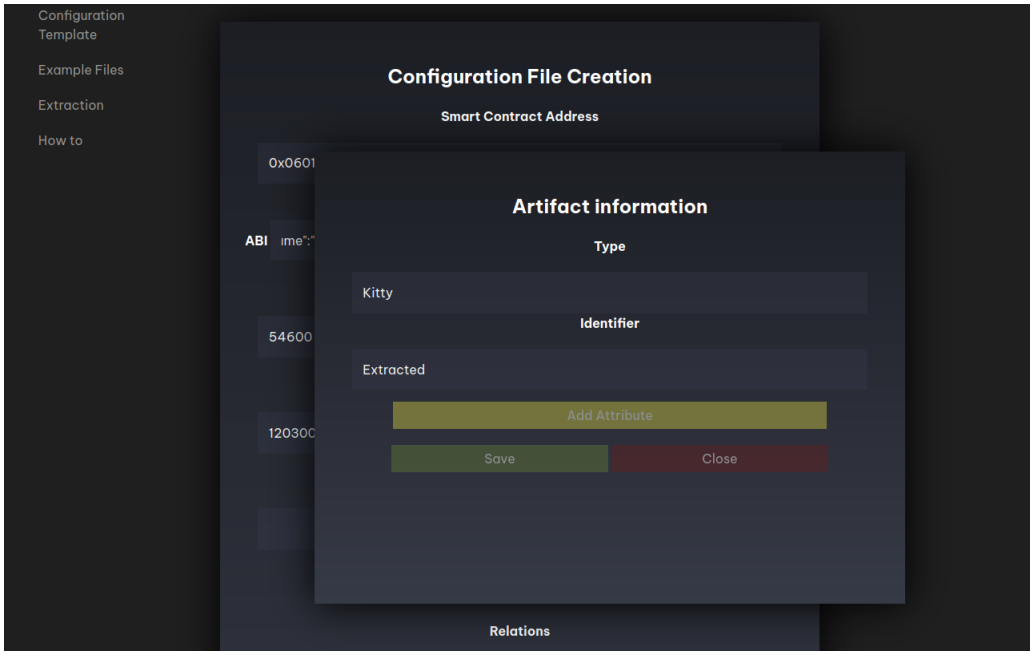

Figure 5.6: Configuration module: Artifacts

event as well as the names of its parameters (see Fig. 5.7).

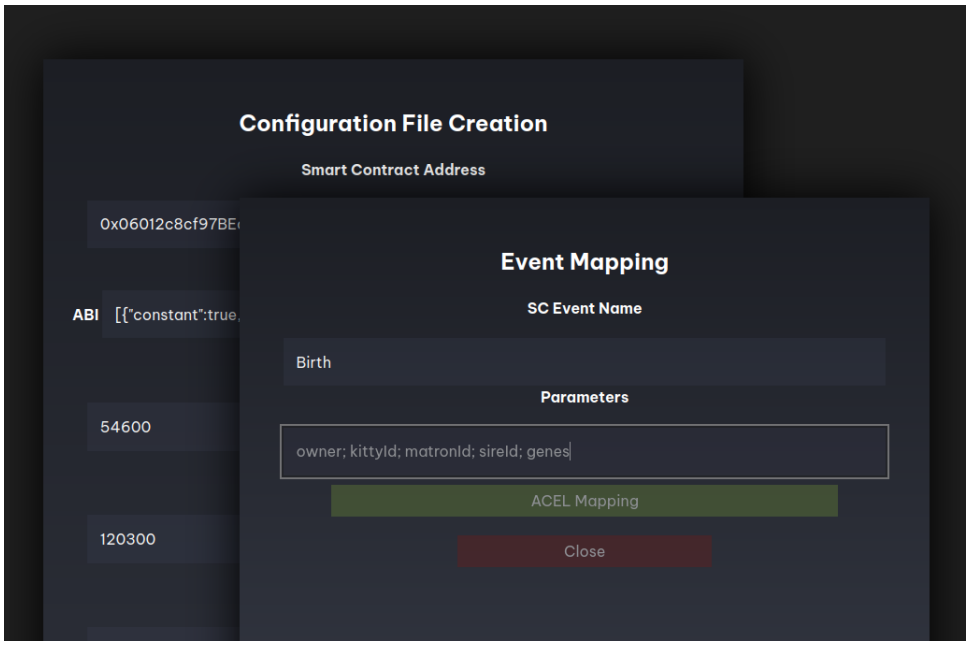

Figure 5.7: Configuration: Smart contract event topic

To define mapping rules, he needs to specify the name (activity name) of the ACEL event (see Fig. 5.8) and select each element from the list of artifacts and relations to define their mappings.

For each selected element a form for its attributes will appear to allow the user to select for each attribute its corresponding value from the list of parameters (see

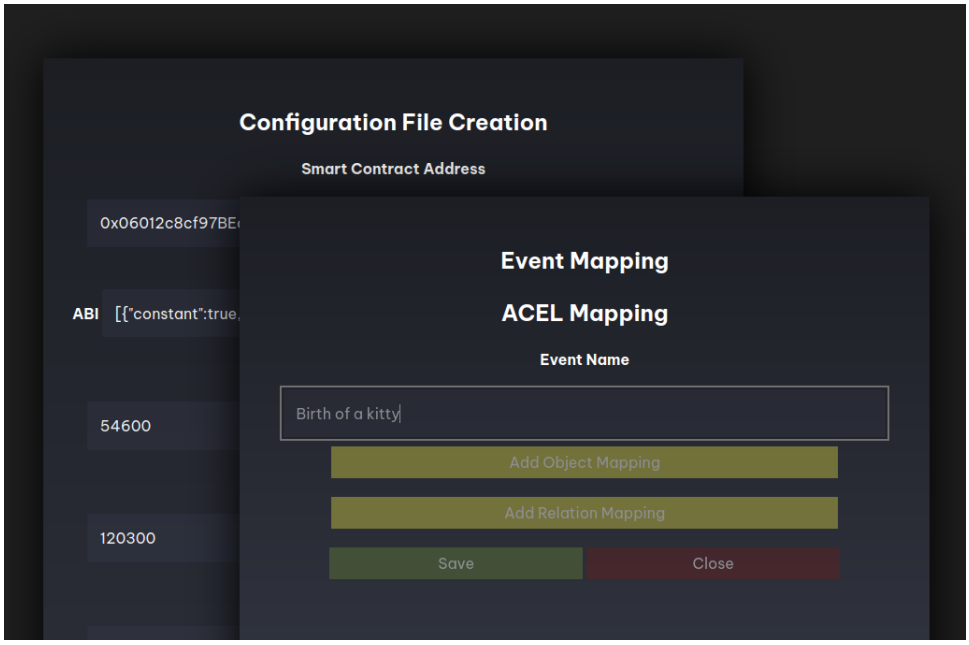

Figure 5.8: Configuration: ACEL Event mapping

Fig. 5.9). Once all events are created a click on the Save File button will generate and automatically download the configuration file, which is formatted in JSON.

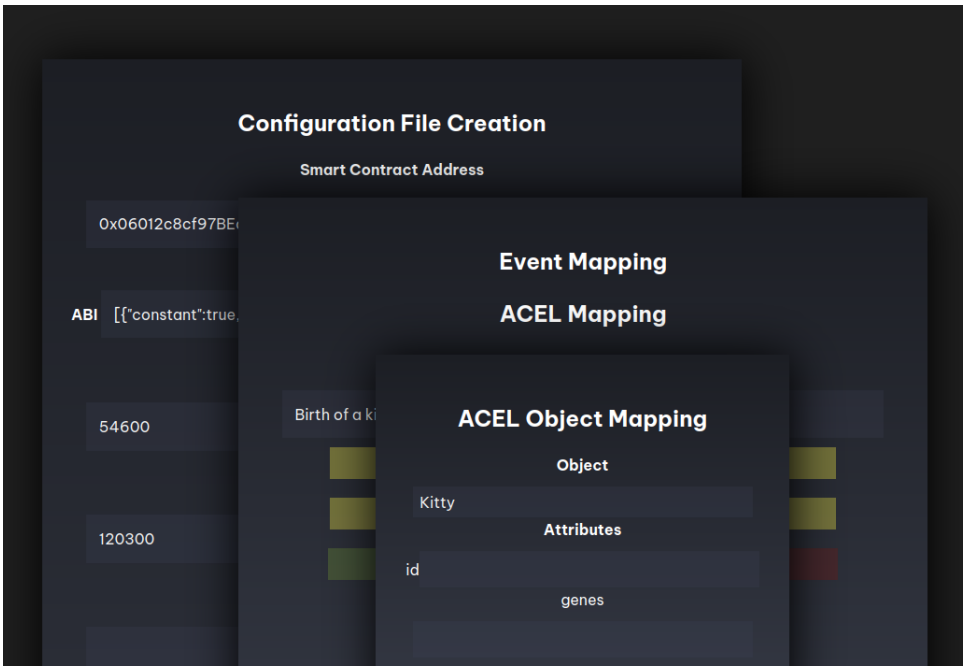

Figure 5.9: Configuration module: ACEL Object Mapping

Extraction module This module takes as input a JSON-formatted configuration file and automates the process of extracting Event data as ACEL elements and generating ACEL logs. To test the extraction module, the user needs to click on

the Extraction option from the left menu. A form to upload the configuration file will appear. The user can then browse to select a file and click on the Generate ACEL Log button to start the extraction process (see Fig. 5.10), after which an ACEL log is automatically downloaded. To test this module, users can use one of the examples files provided via the link `https://lazy-jade-armadillo-cape.cyclic.app/upload/#files`.

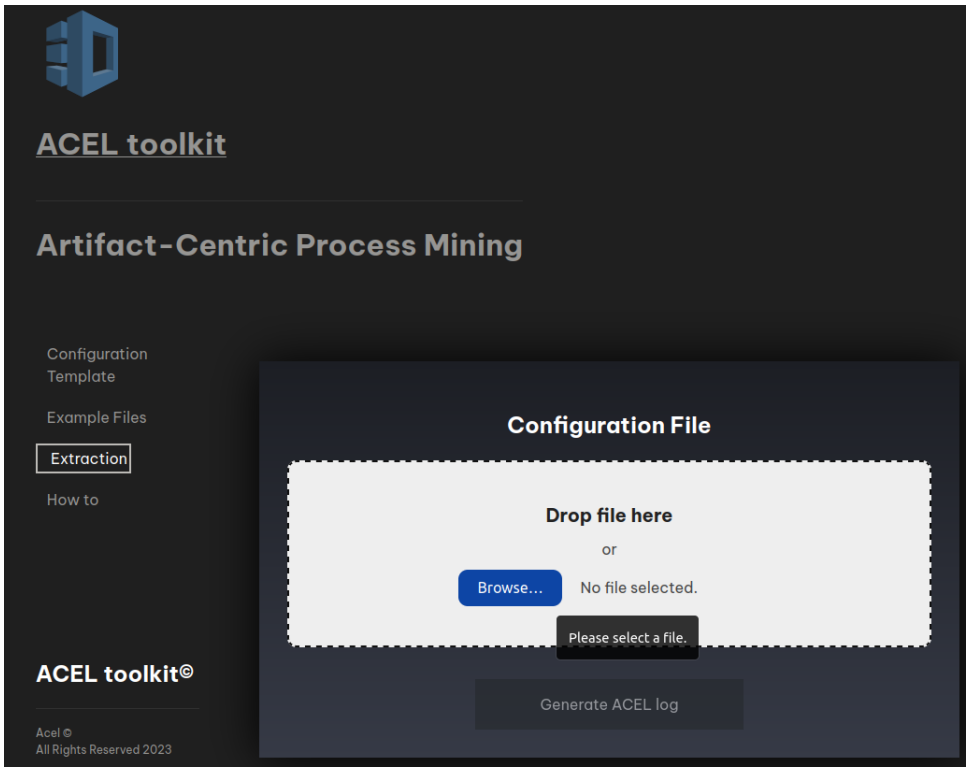

Figure 5.10: Extraction module

Execution scenario The extraction and generation phases are automated and thus are not visible to the user. In the following, we present an execution scenario from start to end to better illustrate how our approach can be used to generate ACEL logs for the motivating example (see Section 1.2). Initially, users complete a form (refer to Fig. 5.5), serving as a template, to produce a configuration file. A simplified representation of the resulting configuration file, based on the cryptokitties example, is depicted in Table 5.1. This table showcases the user-provided details necessary for the system to generate event data associated with the Birth event. Leveraging the provided Smart Contract Reference, the system retrieves the corresponding smart contract events from the blockchain and reverse hashes them using the ABI to extract their topics and parameter values as illustrated in Table 5.2. The smart contract events topics and parameter values serve as the basis for generating an ACEL log, guided by the mapping rules outlined in Table 5.1.

Table 5.1: Configuration file

SC Reference	**Artifacts**	**Relations**
Address=0x06012c.. startBlock=11115450 endBlock=11115550	kitty(id:static, genes:static owner:dynamic..)	hasMother (source:cat, target:cat..) hasFather(source:cat, target:cat..)

(a)

Event Data Mapping			
Sc Event Topic	**Event mapping**	**Object mapping**	**Relation mapping**
Birth(owner, kittyId, matronId, sireId, genes)	Name=Birth timestamp= block.timestamp resource=owner	(type=cat, id=kittyId, genes=genes, lifecycle=born) (type=cat, id=matronId, lifecycle=becameMother) (type=cat, id=sireId, lifecycle=becameFather)	(type=hasMother, source=kittyId target=matronId, changeStatus= addedTarget) (type=hasFather, source=kittyId, target=sireId, changeStatus= addedTarget)])

(b)

Table 5.2: Smart contract event

	Topic	**Data**
Hashed	0x0a5311bd2a...	0x00000...000d90909..
Preprocessed	Birth(owner, kittyId, matronId, sireId, genes)	owner = 0xD9090.. kittyId = 1971388 matronId = 1806834 sireId = 1279559 genes = 46400655842..

5.5.1.2 Evaluation and discussion

To confirm our qualitative assessment of the performance of ACEL (Section 4.4) compared to other formats, we extracted XES and OCEL logs from the motivating example Cryptokitties (Section 1.2). In this evaluation we did not use XOC logs as the approach for extracting XOC logs used in [16] is only adapted to relational databases and cannot be used on blockchain data. We extracted a restricted event subset (blockRange from 11115450 to 11115550) to facilitate log comparison. However, the subsequent extraction included all the smart contract event types found in the source code of the application's smart contracts. To extract the XES log, we followed the approach of Klinkmuller et al. [19], which provides a configuration file they call manifest [11] and a tool [12] to extract XES logs from Ethereum. We used our tool to generate the ACEL log. Finally, we adapted our implementation to generate OCEL logs. The comparative count of events, relations, objects and traces found in the three logs (i.e., XES, OCEL and ACEL) is listed in Table 5.3.

A straightforward comparison among the three types of logs (Table 5.3) reveals notable quantitative distinctions. ACEL stands out by capturing 34 relationships

[11] `https://ingo-weber.github.io/dapp-data/`

[12] `https://github.com/ChrisKlinkmueller/Ethereum-Logging-Framework`

Table 5.3: Comparison of XES, OCEL and ACEL logs.

	XES	OCEL	ACEL
Events	218	184	184
Objects		218	194
Relations			34
Traces	194		194

among kitty instances, a detail lost in other log formats. Regarding the traces, the XES and ACEL logs have the same number of traces which is 194 traces. The number of traces (194) in ACEL has been computed by simply counting the number of objects. The events associated to each object can then constitute one trace. Moreover, ACEL stores fewer objects (194 compared to OCEL's 218), while XES does not retain this information. The XES log exhibits a higher count of events (218 compared to 184 in ACEL and OCEL logs) because each object change is identified as an event within the provided sample manifest by the approach's authors [19].

It's important to note that some events listed in the sample manifest, used to extract the XES log, do not directly correspond to actual activity executions within the cryptokitties smart contract. Instead, they represent relations between kitties, a facet captured explicitly by ACEL. For instance, the Birth event, signifying one activity in the cryptokitties application, results in the birth of a kitty along with the introduction of parental relations (sire and matron). In the manifest, this single event is extended into three events (Is Born, Give Birth as Matron, Give Birth as Sire). These additional events are the only mean to capture information about occurring changes and interactions. This speaks to the inadequacy of XES in capturing accurate and complete artifact-centric event data. There will always be the need for a tradeoff between accuracy and completeness. Conversely, ACEL represents this as a single event (Birth of a kitty), altering three kitties' lifecycles (new states: born, becameFather and becameMother) and introducing two relations (hasFather and hasMother) linking the born kitty to its parents. This representation in ACEL seems to offer a more realistic view of the cryptokitties process as it does not introduce new executed activities but effectively captures their impact on evolving objects throughout the process.

Finally, it is important to note that all the information present in XES and OCEL is also present in ACEL and with some filtering we can obtain XES or OCEL logs from an ACEL log. ACEL logs on the other hand are richer since they contain information regarding the evolution of objects and their relations. Thus, this quantitative evaluation confirms the qualitative evaluation conducted in section 4.4.

5.5.2 Process mining on ACEL logs

In this section we evaluate our approach of adapting ACEL logs to existing process mining techniques, in particular discovery techniques. We consider the technique to discover OC-DFGs used in [14] and we apply it to two Ethereum applications as case studies, namely the motivating example Cryptokitties and Augur [13], a prediction market platform. We use the Pm4py [14] implementation of the OC-DFG discovery algorithm with ACEL-filtered OCEL logs for both Cryptokitties and Augur.

5.5.2.1 Case study 1: Cryptokitties

As discussed in Section 1.2, Cryptokitties is an Ethereum-based application focused on the acquisition and breeding of virtual kitties through auction mechanisms. Within this context, both the kitty and the auction serve as artifacts that evolve as a result of various engagements from users, e.g., breeding and auction creation. To conduct our analysis, we initially extracted an ACEL log from the blockchain data pertaining to Cryptokitties. Subsequently, this ACEL log underwent filtering to derive an ACEL-filtered OCEL using our filtering technique (detailed in Section 5.4.2). Concurrently, we extracted a classical OCEL from the identical blockchain dataset. We then used the Pm4py tool to discover OC-DFGs from those two logs. The outcomes are visually represented in Figure 5.11, which shows the result of applying the discovery technique on the classic OCEL log, and in Figure 5.12, which shows the result of discovery using the ACEL-filtered OCEL log.

5.5.2.2 Case study 2: Augur

Augur operates as a decentralized platform for predicting future events, constructed on the Ethereum blockchain. Within Augur, users can engage in markets foreseeing verifiable real-life events, placing bets on their potential outcomes and receiving payouts upon market resolution. For instance, a market like *"Will the price of Ether exceed 10,000 by August 31, 2024?"* offers users two options to bet on: *'yes'* or *'no'*. The resolution of a market occurs after its designated 'end event' takes place, and the reported outcome is provided by the initial market reporter. This reporter, selected by the market creator upon its creation, holds the responsibility to declare the market outcome post the *'end event'*. The initially reported outcome can be challenged, allowing other reporters to dispute it by proposing a new outcome. Users

[13] `https://augur.net/`
[14] `https://pm4py.fit.fraunhofer.de/`

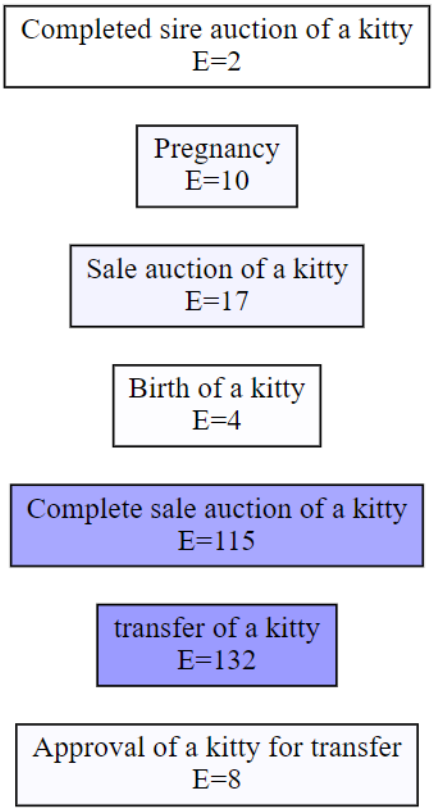

Figure 5.11: OC-DFG of the Cryptokitties classic OCEL log

can support these new outcomes through crowdsourcing [15]. The outcome receiving the most support becomes the final outcome, concluding the market. Similarly to our approach with Cryptokitties, we extracted Augur's events from Ethereum and generated an ACEL log. Employing our filtering technique, we derived an ACEL-filtered OCEL log from the initial ACEL log. Utilizing this filtered log with the Pm4py tool, we generated an Object-Centric Directly-Follows Graph (OC-DFG), depicted in Figure 5.13. This OC-DFG illustrates four primary artifacts: Market, Universe, Order, and Token.

5.5.2.3 Discussion

The objective of the previous case studies was to evaluate the advantages of ACEL over OCEL when it comes to process mining in the context of artifact-centric processes. The primary advantage of ACEL over OCEL in the context of artifact-centric processes becomes evident upon comparing the representations depicted in Fig. 5.11 against Fig. 5.12 and Fig. 5.13, where the first figure represents the list of activities

[15]Crowdsourcing in the context of Ethereum's Augur platform refers to the decentralized, collective effort of individuals (often referred to as "reporters" or "participants") to contribute to the designation of the right outcome by using their assets to vote for it.

of the process and the other two figures depict actual OC-DFGs illustrating object and event interrelations. Indeed, the discovery technique was not able to discover an OC-DFG from the classic OCEL log as each object was affect by one activity and thus the discovered model is just a list of activities with their event count. On the other hand, the ACEL-filtered OCEL logs allows the discovery of OC-DFGs each object in these logs was affected by many events. This disparity arises due to the fundamental distinction in log structures: classic OCEL logs generate new objects for each object change, contrasting with ACEL logs, which support object evolution without necessitating the creation of new objects. This results in the case of OCEL in a log where each object is linked to only one activity and therefore no DFG can be discovered. This distinction underscores ACEL's suitability and consequential enhancement in process mining outcomes, particularly in applications where objects undergo evolution, as observed in artifact-centric processes. Fig. 5.12 and Fig. 5.13 demonstrate that the OC-DFG derived from ACEL logs encapsulates multiple perspectives (all case notions) and their interactions within a unified model. Each perspective represents the process model associated with a distinct case notion or artifact. For instance, in Fig. 5.13, the consistent linkage between the market perspective and the universe perspective via the activity Creation of a market is apparent. Moreover, these visualizations give a glimpse into the lifecycle of each artifact, as they outline the associations of objects with pertinent activities.

Consequently, insights into the behavioral tendencies of the application's artifacts emerge (given by most likely paths), such as the likelihood of a dispute being initiated but not concluded while the initial reporter receives compensation for their service.

This particular behavioral pattern may suggest underlying factors, such as the efficiency of incentives (Augur's reward and punishment system is working as reporters are correctly reporting) or user engagement (users are not incentivized to conclude their dispute), though the definitive elucidation rests with domain experts. This insights points the domain experts to an abnormal behavior which may have many interpretations. Nonetheless, the OC-DFG serves as a comprehensive framework aiding the decision-making process for domain experts, offering a consolidated representation encompassing all perspectives simultaneously. Within this map lie valuable insights into user behavior and data perspectives, facilitating informed decisions by domain experts.

5.6 Conclusion

In this chapter, we answered the research question raised in Section 1.3, which is: $RQ2$: How to collect artifact-centric event data from blockchain applications ?. We proposed an approach to collect artifact-centric data from blockchain applications and automatically generate ACEL logs from this data. We divided the approach into three phases: a configuration phase (to gather required domain knowledge), a collection phase (to collect event data from blockchain), and a mapping and generation phase (to map collected event data into ACEL elements and generate ACEL logs). We designed a configuration template to allow for an easier and valid configuration process. We automated the collection and mapping & generation phases through two algorithms that we implemented in a tool available online. This tool contains two modules: a configuration module which helps the user generate a configuration file, and an extraction module which given a configuration file generates the corresponding ACEL log. We evaluated the applicability of our tool on the Ethereum application Cryptokitties and the resulting log allowed us to compare ACEL to other formats and confirm its performance qualities.

We also sought to test the perspective of ACEL for process mining using logs generated by our tool. The tests were conducted on two Ethereum applications, namely CryptoKitties and Augur. We showed that ACEL logs with some filtering can be used with object-centric process mining techniques, which support OCEL, to discover artifact-centric models. We proposed a filtering approach to generate OCEL logs from ACEL one while keeping object evolution. We generated ACEL logs for each application and filtered them to obtain OCEL logs. We tested the discovery of OC-DFGs from both types of logs. Evaluation results showed the benefits of ACEL over OCEL for the discovery of artifact-centric models. They also showed examples of insights, such as the detection of abnormal behaviors in the discovered models. This first evaluation of the potential of ACEL for process mining motivates the necessity to propose a process mining technique adapted to ACEl in order to take full advantage of its potential.

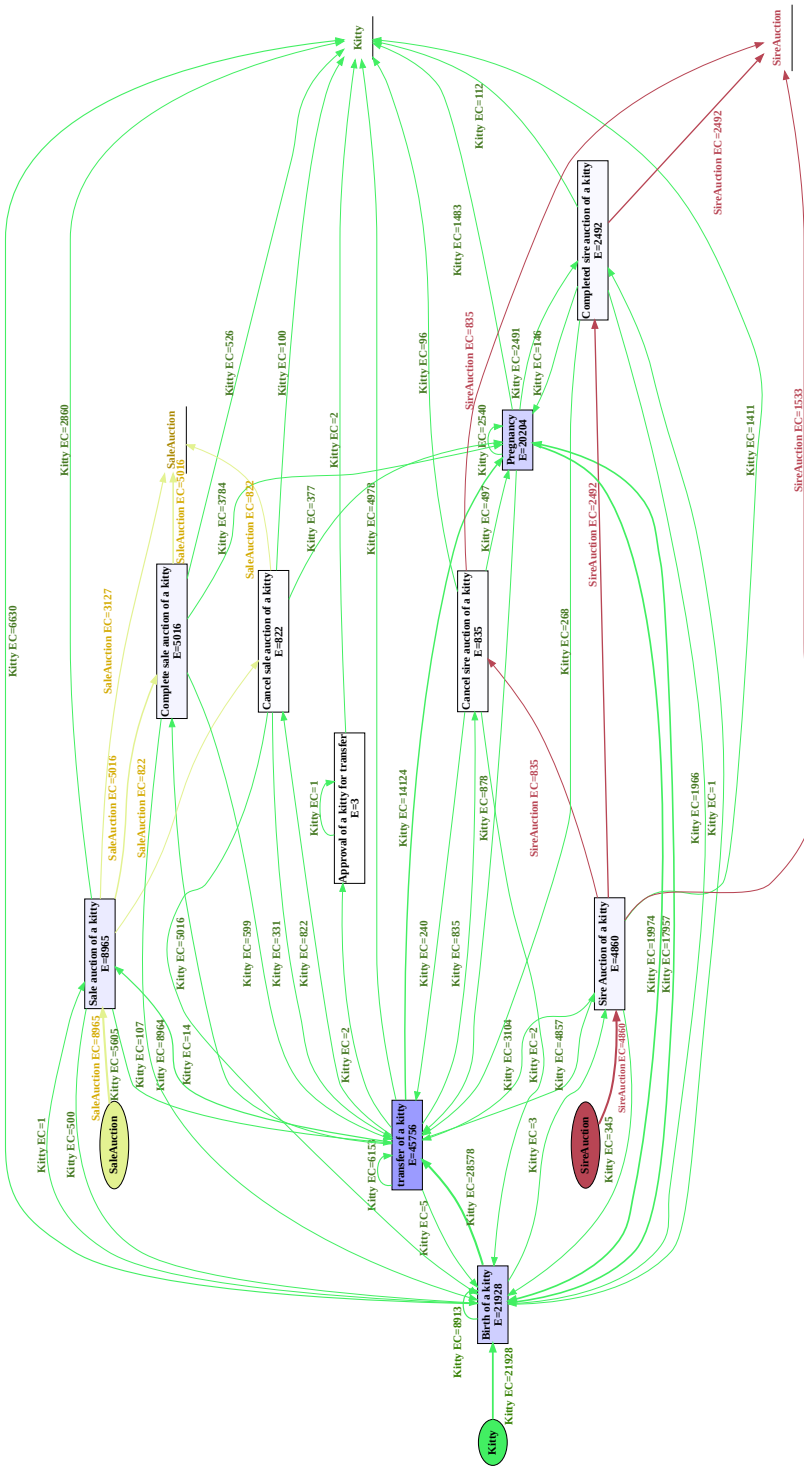

Figure 5.12: OC-DFG of the Cryptokitties ACEL/OCEL log

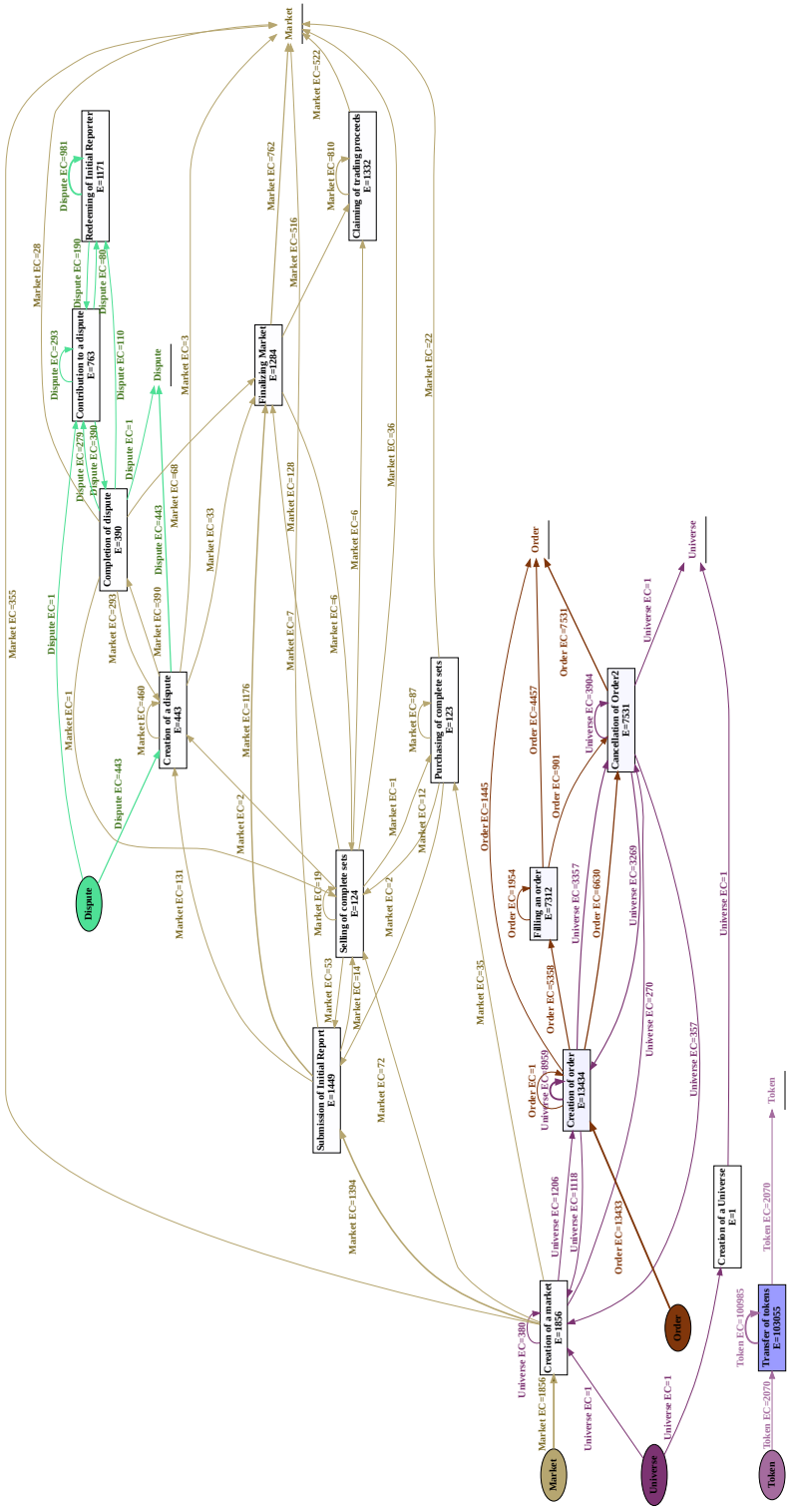

Figure 5.13: OC-DFG of the Augur ACEL/OCEL log

Chapter 6

Artifact-Centric Process Mining for Blockchain Applications

Contents

6.1 Introduction

This chapter presents our approach to discover GSM models from artifact-centric event logs and thus answer $RQ3$ (How to discover artifact-centric process models from artifact-centric event logs ?). To do so, we propose a discovery approach

based on hierarchical clustering and that does not rely on domain knowledge nor translation mechanisms. This approach uses invariants detection [67] to discover data conditions and information gain [68] of common data conditions to cluster activities into nested stages to unveil the hierarchical structure within each lifecycle.

We start this chapter by providing background on information gain and hierarchical clustering in Section 6.2. We then outline our approach and the design choices made in Section 6.3. Subsequently, the details of the different phases of our approach are provided in Section 6.4. Finally, Section 6.5.1 presents the implementation of our approach and its evaluation using our motivating example (Section 1.2) as a case study.

The work in this chapter was published in the proceedings of the CoopIS conference [36].

6.2 Preliminaries

6.2.1 Information Gain

Information gain ($\mathcal{IG}$) is a concept commonly used in the field of machine learning and decision tree algorithms [1], particularly in the context of feature selection [2] to decide which feature to split the data on at each step in the decision tree. $\mathcal{IG}$ measures the uncertainty about a target variable after partitioning the dataset based on a feature, thus quantifying how effectively labeling the dataset by that feature can help predict the target variable. Mathematically, $\mathcal{IG}$ is calculated using concepts from information theory, specifically entropy [69]. Entropy measures the impurity or uncertainty in a dataset, i.e., the unpredictability of its labels. The lowest entropy (zero) signifies a dataset where all elements possess the same label, resulting in a state of complete homogeneity (a pure dataset). Conversely, the highest entropy (one) characterizes a dataset with equal proportions of different labels across its subsets, leading to a state of maximal heterogeneity. $\mathcal{IG}$ operates inversely to entropy, wherein an entropy of zero corresponds to an $\mathcal{IG}$ of one. A higher $\mathcal{IG}$ value for a feature indicate that a feature provides more substantial information for predicting a label, whereas a lower $\mathcal{IG}$ value signify less useful information in determining the label from that particular feature. When a dataset is split based on a feature, the

[1] Decision tree algorithms create a model that predicts the value of a target variable based on several input variables by splitting data into subsets based on the value of those input variables, forming a tree-like structure of decisions.

[2] Feature selection in the context of decision trees involves identifying and selecting the most informative feature or variables from a dataset that contribute significantly to the prediction outcome.

information gain is the difference between the entropy of the original dataset and the weighted average of the values of entropy of the resulting subsets [68]. The formula for information gain often used in decision trees for a variable v and a dataset A is:

IG (v, A) = Entropy before split − Weighted average of entropies after split

6.2.2 Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters [70]. In the realm of data analysis, it is employed for the grouping of similar data points of a dataset into clusters, where it uniquely constructs a tree of clusters known as a dendrogram. Hierarchical clustering is categorized into two primary methodologies: agglomerative and divisive. Agglomerative, or bottom-up clustering, begins with each point as a distinct cluster and merges them step by step based on their similarity, visualized through a dendrogram. Conversely, divisive or top-down clustering starts with all points in one cluster and divides them recursively. Hierarchical clustering groups or divides data points based on their similarity or dissimilarity, where the choice of similarity measure can include distance metrics such as Euclidean, Manhattan, or domain-specific measures. In the agglomerative approach, the algorithm treats each data point as an individual cluster initially, then iteratively merges the nearest pair of clusters until all points are unified into a single cluster or until achieving a specific stopping criterion. The clustering requires the use of a distance matrix to measure the dissimilarity between clusters, determining which clusters to merge based on a linkage criterion. The linkage criterion, such as single, complete, or average linkage, defines the distance between two clusters. Single linkage uses the minimum distance between points in two clusters, complete linkage uses the maximum distance, and average linkage uses the average distance. The divisive approach, starting with all points in one cluster, recursively splits the most diverse cluster until reaching individual data points or a predefined stopping point [71].

6.3 Approach Overview

Discovering the GSM model of an artifact-centric process, like the one depicted in Section 2.3.1, encompasses the discovery of two elements. First we need to discover the **information model** of every artifact relevant to the process, then we need to discover the **lifecycle of each artifact**. The information model is composed of the artifact attributes and the lifecycle comprises, the different stages and their

Figure 6.1: Overview of the GSM models discovery approach

hierarchical structure, as well as the guard(s) and milestone(s) of each stage, and finally, the interactions between the different artifacts.

Discovering an artifact's information model from an ACEL log amounts to a simple extraction process due to ACEL's storage of the artifact relational model. This extraction process involves gathering attribute names for each artifact and incorporating foreign keys into the information model based on the relational connections between artifacts.

Consequently, we chose to focus on the discovery of the lifecycle, which essentially revolves around discovering stages' guards, their interactions, and their hierarchical structure. Fig. 6.1 provides an overview of our proposed approach, where we divide it into two phases: (i) the discovery of guards and interactions; (ii) the discovery of stages and nested stages. In the following, we will explain how the first phase sums to the discovery of data conditions and the second phase can be achieved through hierarchical clustering.

The discovery of guards is a fundamental prerequisite for discovering stages and delineating their hierarchical structure. A guard comprises a sentry encapsulating an internal or external event, e.g., in the GSM model of the motivating example in Section 2.1.2.1 (k.'Pregnant'.achieved()) and (k.'giveBirth'.onEvent()), and data conditions e.g., (k.'cooldown'$\leq$ currentTime). Notably, in the context of ACEL, where external events are absent, our focus narrows to internal events for guard sentry discovery. These internal events pertain to milestone achievements, stored in ACEL as values of the object attribute lifecycle. Therefore, the identification of these internal events aligns with the discovery of data conditions. In the context of GSM, internal events are expressible through data conditions, e.g., (k.'Pregnant'). Thus, discovering guard sentries, encompassing internal events and data conditions, becomes a challenge of discovering data conditions related to artifact attributes and their milestones.

The milestones attributed to a stage may encompass some or all milestones of its sub-stages or introduce new milestones. In the case of an atomic stage, the milestones correspond to those of its associated task or activity. Notably, the milestones of the parent stage may not necessarily align with the milestones of the task. However, the notion of task-independent milestones is not supported within ACEL.

Instead, in ACEL, an activity (task) is linked to one lifecycle (milestone) change, since it is not tailored to the nuances of GSM. Introducing stage-specific milestones could be achieved through an optional custom object attribute [35]. Yet, given our reliance on traditional ACEL logs, we opt not to introduce stage-specific milestones, defining the milestones of a parent stage as the conjunction of the milestones its atomic sub-stage.

Within GSM, the interactions among artifacts are conveyed through internal events concerning the attainment of milestones by other artifacts [72]. These events are discovered through the discovery of data conditions, as previously mentioned.

The discovery of GSM stages hinges solely upon identifying their guards, as each stage is uniquely identified by its set of guards. Notably, two stages cannot share identical guards. Additionally, sub-stages possess their distinct guards while also inheriting the guards of their parent stage. This inheritance establishes a hierarchical relationship between sub-stages and their parent stage. For instance, in the motivating example, the Procreation stage serves as the parent of the Breeding and Birth stages. This hierarchical representation suggests that the parent stage acts as a cluster of sub-stages, each of which can further function as an independent cluster housing its own sub-stages. Consequently, modeling the discovery of stages and their hierarchical arrangement aligns with a hierarchical clustering problem wherein the similarity is based on common guards. Importantly, these clusters must maintain loose coupling, ensuring that the guards of one stage do not inadvertently trigger the activation of another stage.

6.4 Discovering GSM models from ACEL logs

The following sections detail the different phases of our approach to discover GSM models: **discover guards and interactions** as data conditions (Section 6.4.1) and clustering activities to **discover stages and nested stages** (Section 6.4.2).

6.4.1 Discovering Guards and Interactions

In accordance with the insights provided in Section 6.3, the process of discovering guards and interactions using an ACEL log can be assimilated to the discovery of data conditions. These data conditions represent properties verified by observed attribute values preceding the execution of an activity. They bear resemblance to invariants [3] that remain true at specific program points [73]. Hence, the method-

[3]Invariants in program analysis are conditions or properties that remain constant throughout the execution of a program or within specific portions of the program.

ologies utilized to dynamically identify likely program invariants can be applied to discover data conditions within a process, as exemplified in [74]. We have adapted and tailored the latter approach to suit our specific context.

In the following, we delve into the details of how to obtain data conditions from an ACEL log using invariants detection systems. Section 6.4.1.1 shows how to generate the input of these systems, i.e., data traces which are lists of attribute values prior to an activity's execution. Subsequently, in Section 6.4.1.2 we present how to apply invariant detection mechanisms to discover the data conditions prevailing within these data traces.

6.4.1.1 Data Traces

A data trace serves, in the context of program likely invariants detection, as a repository containing the variable values at specific points within a program. When considering a business process (BP), a data trace encapsulates the variable values either preceding or succeeding the execution of an activity within the process.

Systems such as Daikon [73] specialize in generating data traces for a program by analyzing its source code. However, in the case of a business process, these traces are extracted from the BP event logs instead of being generated from source code.

The approach outlined in [74] delineates a methodology for extracting data traces within the context of a business process. This extraction method involves the systematic replay of events pertaining to a process instance against a reference model. This replay process serves to update the values associated with each variable, ultimately capturing a single data trace. This procedure halts its execution before reaching the targeted activity, thereby providing a business process data trace.

In our approach, we propose an innovative approach aimed at extracting data traces from an ACEL log with no dependency on a reference model. We start by giving in Section 6.4.1.1.1 our definition of artifact-centric process instances, one that does not cause convergence and divergence and allows the discovery of interactions. We also define data traces in our context, in Section 6.4.1.1.2, as being specific to each activity. Finally, we detail in Section 6.4.1.1.3, the mechanism we use to generate data traces from artifact-centric process instances.

6.4.1.1.1 Artifact-centric process instances. Our approach involves defining a process instance for each artifact's lifecycle to facilitate the discovery of stages associated with each artifact. Since our objective extends to the discovery of interactions between artifacts, we also consider events linked to related artifacts. Therefore, we propose a simple definition for a process instance:

Definition 6.4.1 (Artifact-centric process instance) *An artifact-centric process instance comprises the sequence of events associated with an individual artifact instance and its related instances until the end of their relations.*

The identification of ACEL events concerning related artifact instances through their relations becomes viable due to ACEL's capacity to capture the evolution of these relations. This process is accomplished utilizing the `changeStatus` attribute associated with the relation's `target`, wherein the value `'deletedTarget'` signifies the end of a relation. The end of a relation marks the end of an interaction between two artifact instances, consequently minimizing the probability of irrelevant data conditions.

For example, considering a scenario involving a breeding event between a pregnant kitty ($o1$) and a prospective father kitty ($o2$) linked through a `breedingWith` relation. Following a birth event, this relation is deleted. While $o2$ remains in a `breedingWith` relation, it is impacted by $o1$'s events, anticipating a subsequent birth event for $o1$ before being allowed to breed again. However, post-termination of the relation, $o2$ ceases to be influenced by $o1$'s events. If, subsequent to the end of the relation, $o1$ engages in breeding with a third kitty, it has no bearing on $o2$.

Our methodology also accounts for scenarios where artifact instances persist without "dying" and the log records multiple iterations of an artifact's lifecycle. Previous methodologies, as discussed in [8], predominantly address instances where artifacts undergo a single lifecycle iteration ($\mathsf{created} \rightarrow \mathsf{updated} \rightarrow \ldots \rightarrow \mathsf{terminated}$). In contrast, our approach accommodates instances where artifacts revisit lifecycle stages multiple times or indefinitely, such as a kitty's perpetual ability to engage in breeding activities.

6.4.1.1.2 Artifact-centric activity specific traces.

Building upon the previously established definition of an artifact-centric process instance and acknowledging the potential for artifacts to revisit lifecycle stages, we formulate a definition for an activity-specific trace:

Definition 6.4.2 (Activity-specific trace) *An activity-specific trace is a sequence of events, within a process instance, delimited by two events related to the targeted activity.*

Notably, the events resulting from the execution of the activity itself are excluded from the activity-specific trace. This deliberate exclusion is necessitated by our objective to discover data conditions leading up to the activity's execution. Therefore, the activity-specific trace encapsulates a chronological sequence of events from a

process instance, commencing after one execution of the targeted activity and concluding before its next occurrence. The steps required to obtain activity-specific traces are listed in Algorithm 3.

The Algorithm uses two steps to generate activity-specific traces. First, we start constructing each activity's traces by creating a trace for each object instance [4] affected by an event of this activity (line 13). In each trace, we add the other activities' events that affect the object instance and are situated between two events of the activity (line 14), i.e., the activity's events are not added. Second, we add to each trace the events which affect object instances related to the trace's main object instance (its identifier). To reduce the search we use a window of related object instances' events (line 26) between a start event (line 20), which corresponds the first activity's event preceding the trace, and an end event (line 21), which corresponds the activity's event that directly follows the trace. We only include the events of related instances that happen before the end of the relation linking the related object instances and the main object instance (lines 27-29).

6.4.1.1.3 Data Trace Generation. To derive a data trace from an activity-specific trace, we rely on a reverse traversal of the events, extracting data values without replaying the events, described in Algorithm 4.

Beginning from the final event (line 9), we collect the first encountered value for each attribute (lines 11, 13, 16, 20, 23) associated with every artifact instance and we name it using a specific namespace (lines 14, 17, 21, 24). This namespacing of attributes serves multiple purposes: enhancing readability, quantifying artifact interactions, and shedding light on potential new interaction types. The namespacing is as follows: In GSM, related artifacts are referenced within the information model of the main artifact, i.e., they are related to, through a foreign key attribute. However, this attribute's name might lack explicit information regarding the artifact's type. Consequently, we augment this attribute's name in the data trace by appending the name/type of referenced artifact. For instance, as depicted in the motivating example, the attribute `k.siringWithId` references another kitty. To explicitly specify this reference, we prefix it with its artifact type, resulting in `k.kitty.siringWithId` (lines 14, 17).

Contrarily to GSM, within ACEL, artifacts relations are stored as separate elements, removing the need for foreign key attributes. Consequently, we extend the previous namespace by incorporating the relation's type as another prefix within the data trace (`k.breedingWith. kitty.siringWithId`) (lines 21, 24). Through

[4] Object instances are the identifiers of activity traces.

Algorithm 3: Generation of activity-specific traces

Data: acelLog $< E, O, R >$, E set of events, O set of objects and R set of relations. $OT \leftarrow \emptyset$, set of object types and $A \leftarrow \emptyset$, set of activities.

Result: AT, a function associating to each couple (activity, object) a set of traces.

Let *OTA be a function whose domain is OT* $\forall ot \in OT$, $\underline{\exists}(a_1, ..., a_n) \in A^n, OTA(ot) \leftarrow (a_1, ..., a_n)$.

Let *OI be a function whose domain is OT* $\times$ *A* $\forall ot, a \in OT, A$, $\underline{\exists}(o_1, ..., o_n) \in O^n, OI(ot, a) \leftarrow (o_1, ..., o_n)$.

▷ Step 1 adding events linked to each main artifact instance ;

ForEach *e of E*
- a ← activityName(e) ;
- **ForEach** *o of objectList(E)*
  - ot ← type(o) ;
  - **ForEach** *act of OTA(ot)*
    - **If** *a == act*
      - Close last set of $AT(ot, a, o)$;
    - **else**
      - **If** *last set of* $AT(ot, a, o)$ *closed*
        - Open new set in $AT(ot, a, o)$;
      - Add e to last set of $AT(ot, a, o)$;

▷ Step 2 adding events linked to related instances ;

ForEach *ot of OT*
- **ForEach** *a of OTA(ot)*
  - **ForEach** *oi of OI(ot,a)*
    - **ForEach** Set_{Events} *of* $AT(ot, a, oi)$
      - startEvent ← getPreviousActivityEvent(a, Set_{Events})
      - endEvent ← getNextActivityEvent(a, Set_{Events}) ;
      - **ForEach** *e of* Set_{Events}
        - relatedInstances ← getRelatedObjects(e, oi) ;
        - **ForEach** *ri of relatedInstances*
          - rit ← type(ri) ;
          - $ri_{Set_{Events}}$ ← getEventsInInterval(rit, a, ri, startEvent, endEvent) ;
          - **ForEach** ri_{event} *of* $ri_{Set_{Events}}$
            - **If** *relationNotEnded(*ri_{event}*, ri, oi))*
              - Add e to Set_{Events} of $AT(ot, a, oi)$;

return AT

Algorithm 4: Generation of Data Traces

Data: acelLog $< E, O, R >$, E set of events, O set of objects and R set of relations. $OT \leftarrow \emptyset$, set of object types and $A \leftarrow \emptyset$, set of activities. AT, a function associating to each couple (activity, object) a set of traces.

Result: DT, a function associating to each couple (object type, activity) a set of data traces

```
1  Let OTA be a function whose domain is OT ∀ot ∈ OT,
   ∃(a_1, ..., a_n) ∈ A^n, OTA(ot) ← (a_1, ..., a_n).
2  Let OI be a function whose domain is OT × A ∀ot, a ∈ OT, A,
   ∃(o_1, ..., o_n) ∈ O^n, OI(ot, a) ← (o_1, ..., o_n).
3  ForEach ot of OT
4      ForEach a of OTA(ot)
5          ForEach o of OI(ot, a)
6              ForEach set of Reverse(AT(ot, a, o))
7                  Open new set in DT(ot, a) ;
8                  ForEach e of Reverse(set)
9                      ForEach ob in objectChangeList(e)
10                         ForEach att in objectAttributes(ob)
11                             If ob == o
12                                 If name(att) == 'lifecycle' and ot.'milestone'
                                    not in DT(ot, a)
13                                     Add (ot.'milestone', value(att)) to last set in
                                        DT(ot, a) ;
14                                 else
15                                     If name(att) not in DT(ot, a)
16                                         Add (ot.name(att), value(att)) to last
                                            set in DT(ot, a) ;
17                             else
18                                 If ob in relation with o and relation not ended
19                                     If name(att) == 'lifecycle' and
                                        ot.relationName(o,ob).type(ob).'milestone'
                                        not in DT(ot, a)
20                                         Add
                                            (ot.relationName(o,ob).type(ob).'milestone',
                                            value(att)) to last set in DT(ot, a) ;
21                                     else
22                                         If
                                            ot.relationName(o,ob).type(ob).'milestone'
                                            not in DT(ot, a)
23                                             Add
                                                (ot.relationName(o,ob).type(ob)name(att),
                                                value(att)) to last set in DT(ot, a) ;
24                 Close last set in DT(ot, a) ;
25 return DT
```

this extension, we add business-relevant semantics and elevate the readability of a discovered model. We also use it to indicate the cardinality of interactions, i.e., the number of artifact instances of the same type interacting with a main artifact. Moreover, through this namespacing and its specification of relations' types, we shed lights on a novel interaction type not previously considered in existing literature, to the best of our knowledge. Previous works focused solely on interactions between artifacts. Whereas, in our approach, we consider interactions between instances of the same artifact type, which we call **reflexive interactions**.

To depict interaction cardinalities and reflexive interactions, we consider the Cryptokitties motivating example and specifically the `Birth` stage. A kitty's birth establishes relations between the kitty and both its father and mother. In this scenario, the born kitty acts as the main artifact and engages in two (cardinality) interactions with instances of the same type as the main artifact (reflexive interaction).

6.4.1.2 Data Conditions

To discover data conditions from data traces, our methodology employs the same system for dynamic detection of likely invariants as used in [74]. Alongside data traces, this system [73] necessitates a declaration file associated to each data trace. Declaration files specify the locations, e.g., right before the execution of a function, within a program's execution where data is recorded, along with the variables involved at those points. Within these declaration files, we need to incorporate a crucial attribute known as comparability. Comparability is a signed integer that guides the system in identifying comparable variables. Variables sharing the same comparability value are considered comparable. This attribute aids the system in identifying pertinent invariants, specifically those involving only comparable variables, which in our case correspond to relevant data conditions. In the context of our approach we presume that the comparability is provided.

Our approach to discovering data conditions uses the previously described system for dynamic detection of likely invariants. We use the data traces, generated using the method described in Section 6.4.1.1.3 to create a declaration file for each one. Then, we serve each pair of data traces and declaration files to the system, generating invariants for each activity of each artifact.

6.4.2 Discovering Stages and Nested stages

In our approach to discover nested stages, we rely on common data conditions that possess equivalent information gain ($\mathcal{IG}$). These data conditions with the same $\mathcal{IG}$ will serve as guards to discover stages and their hierarchical structure. Indeed, we operate under the premise that stages sharing a common parent exhibit equal discrimination by the parent's guard when compared to other stages within the artifact's lifecycle. Thus, the parent's guard will have the same $\mathcal{IG}$ every time it is used to differentiate between one of the parent's sub-stages and the rest of the lifecycle's stages. For example, if a stage $s1$, with a data condition dc as its guard, has two sub-stages $s2$ and $s3$, the decision tree which answers the question *Is this stage* $s2$ *? (here* $s2$ *is used as the label)* will have an $\mathcal{IG}$ for dc as feature equal to the $\mathcal{IG}$ of that same feature as another decision tree which answer the same question for $s3$. Essentially, a parent's guard consistently exhibits the same $\mathcal{IG}$ each time it helps separate one of the parent's sub-stages against other stages within the artifact's lifecycle.

The affirmation that guards are data conditions with the same $\mathcal{IG}$ stems from our perspective of considering the parent stage as a label and the data condition as a feature, as explained above in the decision tree example. For instance, in the motivating example, if we label all activities as either `partOfProcreation` or `notPartOfProcreation`, the data condition (`k.'cooldown'` $\leq$ `currentTime`), we can effectively splits the population (i.e., the activities) into two groups of stages. Each activity of the first group would be labelled `partOfProcreation` and each activity of the second group would be labelled `notPartOfProcreation`. Thus, the data condition (`k.'cooldown'` $\leq$ `currentTime`) would serve as the guard for the `Procreation` stage.

In summary, $\mathcal{IG}$ can be used to know which data conditions serve as guards. Moreover, sub-stages inherit the guard of their parent stage. Therefore, the discovery of stages and nested stages consists in: (i) finding each activity's guard by looking for the data conditions that have the highest $\mathcal{IG}$ for that activity and that do not display the same $\mathcal{IG}$ for other activities, (ii) finding the activities which have common data conditions with the same $\mathcal{IG}$ for each activity, and grouping these activities into atomic stages, (iii) iterating the second process to find the parent stages of sub-stages. Thus, stage discovery takes the form of hierarchical clustering, where similarity is based on common data conditions exhibiting identical $\mathcal{IG}$.

Our use of information gain to find data conditions is inspired from the work of de Leoni et al. [74]. In the following, we first position our work from their approach and show how it inspired our work in Section 6.4.2.1. Then, we present the details of our discovery approach: Our proposed similarity function for clustering activities into stages is presented in Section 6.4.2.2, and our approach for hierarchically clustering the stages into nested stages based on that similarity is detailed in Section 6.4.2.3.

6.4.2.1 Limitations of branching conditions for discovering GSM stages

In [74], the authors' utilization of $\mathcal{IG}$ is directed toward discerning conditions that differentiate between two tasks at a branching point in BPMN models [12]. However, this approach does not align with our scenario due to the support for parallelism between stages and activities of the same stage, within the context of GSM models as well as the nesting of stages. Indeed, contrary to BPMN models, in GSM many stages can be opened at the same time and their activities can also be executed at the same time.

The aforementioned work also employs $\mathcal{IG}$ to find to the shortest relevant branching conditions, by retaining only the condition or conjunction of conditions exhibiting the highest $\mathcal{IG}$. For instance, within the `Breeding` stage in the motivating example, if $\mathcal{IG}(!\texttt{k.'Pregnant'}) = \mathcal{IG}(!\texttt{k.'Pregnant'} \;\&\&\; (\texttt{k.'cooldown'} \leq \texttt{currentTime}))$, the condition $(!\texttt{k.'Pregnant'} \;\&\&\; \texttt{k.'cooldown'} \leq \texttt{currentTime})$ would be discarded according to [74]. The discarded condition is the guard of the stage `procreation`, thus its absence hinders the discovery of this stage.

In contrast, our approach refrains from discarding any condition since those with the lowest $\mathcal{IG}$ can be the guards of parent stages. Despite this variance in handling conditions, the utilization of $\mathcal{IG}$ for branching conditions in [74] served as inspiration for our approach to discover stages and nested stages.

6.4.2.2 Similarity Between Activities

In clustering, similarity is quantified using a distance metric, where the proximity between two points within a cluster determines their similarity. Specifically, in our work, we define similarity between two activities based on their common data conditions with identical $\mathcal{IG}$.When two activities possess common data conditions exhibiting the same $\mathcal{IG}$, we consider them to be the closest, indicating a similarity in their nature. The closest possible activities share identical data conditions and identical $\mathcal{IG}$ values for these conditions, rendering their distance measure equal to zero. Conversely, activities that are the farthest apart exhibit either no shared

common data conditions or possess common data conditions with differing $\mathcal{IG}$ values. This disparity in data conditions signifies a complete dissimilarity, thereby yielding a distance measure of one between the activities.

In the following, we define a similarity function inline with these similarity criteria. This function computes the distance between two activities by considering their respective data conditions and their associated $\mathcal{IG}$ values.

Definition 6.4.3 ($\mathcal{IG}$ of an Activity Condition) *Let $\mathcal{A}$ be the set of an artifact's activities, $\mathcal{C}$ the set of these activities' data conditions, $\mathcal{DT}$ the set of all data traces, and $adt\colon \mathcal{A} \longrightarrow \mathcal{DT}$ a mapping associating activities $\in \mathcal{A}$ to their data traces $\in \mathcal{DT}$. The information gain of an activity's condition is defined as:*

$$\forall a \in \mathcal{A}, \forall c \in \mathcal{C}, \mathcal{IG}_a(c) = \mathcal{IG}(adt(a), adt(\mathcal{A} \setminus a), c)$$

The formula states that the information gain of a data condition c for an activity a is equal to the information gain of c when used to discriminate between the traces of a and the traces of all activities minus a.

Definition 6.4.4 (Similarity Function) *Let $a, b \in \mathcal{A}$; C_a (data conditions of a), C_b (data conditions of b) $\subset C$, $CC_{ab} = \{c | c \in C_a \wedge c \in C_b \wedge \mathcal{IG}_a(c) = \mathcal{IG}_b(c)\}$ the set of common data conditions between a and b with the same $\mathcal{IG}$. $n = |CC_{ab}| \forall c_k \in CC_{ab}, k \in \{1 \ldots n\}, \mathcal{IG}(c_k) = \mathcal{IG}_a(c_k) = \mathcal{IG}_b(c_k)$. The distance between a and b is given by:*

$$dist(a,b) = \begin{cases} 1 & |CCab| = 0 \\ 1 - \dfrac{1}{1 + \dfrac{\log \dfrac{|Ca| + |Cb|}{2 \times |CCab|}}{\sum_{k=1}^{n} IG(c_k)}} & |CCab| \neq 0 \end{cases}$$

The formula is designed to measure the dissimilarity between two activities a and b, inversely related to their similarity. It operates under two conditions: (i) When $|CCab| = 0$, indicating no common data conditions with equal information gain, the distance is set to 1, representing maximum dissimilarity. (ii) When $|CCab| \neq 0$, the distance is calculated based on the log ratio of the sum of the sizes of Ca and Cb to twice the size of CCab, normalized by the sum of the information gains of the common conditions. This reflects a lower distance (higher similarity) when there are more common conditions with significant information gains. The highest similarity would be when a and b have the same data conditions with the same information gain, thus the log ratio would be equal to 0 and the formula resolves to 0 (minimum dissimilarity and maximum similarity). The division

by the sum of the information gains of the common conditions guarantees that as the number of relevant common conditions increases (or as their information gain increases), the distance decreases, reflecting increased similarity.

6.4.2.3 Hierarchical Clustering to Discover Stages and Nested Stages

In order to discover stages and nested stages we rely on a hierarchical agglomerative clustering [75] revised to incorporate a customized distance matrix, computed using the similarity function *dist*, and a distinct linkage criterion to determine when two clusters can be merged. In our context, we chose a linkage criterion that merges two clusters only when the distance between all points (i.e., activities) of a cluster with all points of another cluster are identical. To optimize this criterion, we measure the distance between two random activities, one from each cluster. This optimization is possible because all activities of one stage/cluster share the data conditions/guard of the stage and will all have the same distance to any other activity because the similarity is computed using $\mathcal{IG}$ of common data conditions. This distance computation is based on the data conditions shared by all activities within each cluster. Our merging condition dictates that the similarity between two clusters must differ from one. This condition on merging helps ensure that activities lacking common data conditions or possessing common data conditions with different $\mathcal{IG}$ values cannot belong to the same stage.

The clustering process starts with the computation of a distance matrix encompassing all activities within an artifact. During the first iteration, the two closest activities are grouped into a cluster. Subsequently, the distance matrix between the remaining activities and this newly formed cluster is computed based on their common data conditions. The data condition of the newly formed cluster is equal to the common data conditions shared by all of the activities within this cluster. In subsequent iterations, the next two closest activities (or one activity with the preceding cluster if their distance is the shortest) undergo merging. This iterative process persists, necessitating the recomputation of the distance matrix before each iteration and cluster merging, until either a single cluster remains or when the distance between all clusters is equal to one (stopping condition).

Consequently, the outcome of this process yields a hierarchical structure delineating the nesting structure of stages within each artifact's lifecycle.

6.5 Implementation and Evaluation

In the following we present the implementation and the evaluation of our artifact-centric process discovery approach. Section 6.5.1 presents the tool we developed to implement our discovery approach. In Section 6.5.2, we introduce our case study and show the outcomes of applying our tool to this specific case study. Using the previous outcomes, we evaluate in Section 6.5.3 the effectiveness of our approach in terms of accurate discovery results and performance of the discovery technique. Finally, the results of the evaluation and limitations of our approach are discussed in Section 6.5.4.

6.5.1 Implementation

To implement our discovery approach we developed a tool comprised of four Python modules, each executing one of the steps described in Section 6.4. The modules are executed in a sequence, described in Fig. 6.2, where the input of one module is the output of the precedent module. The first module takes as input an ACEL log and the last module gives as output a list of stages with their guards. The sources of the tool are accessible through the link: `https://gitlab.com/disco5/Gsm/-/tree/main/discovery`.

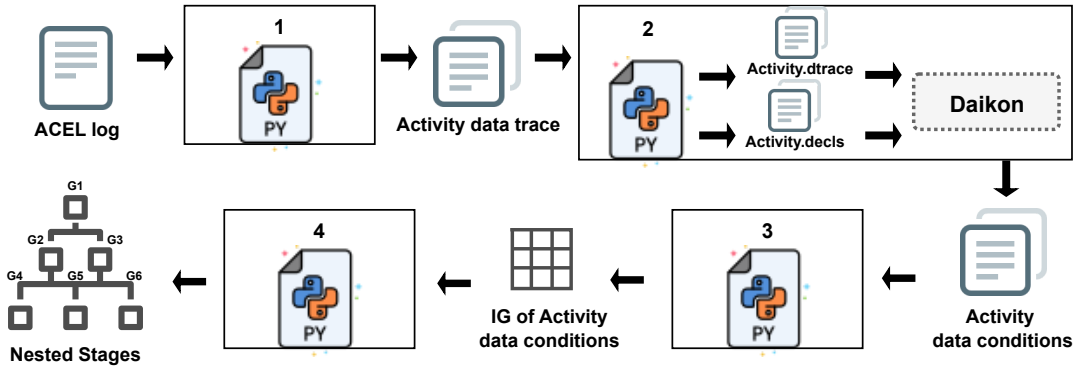

Figure 6.2: Architecture of our discovery tool

The first module operates by taking an ACEL log as input and subsequently generates the activity-specific traces associated with every artifact within the log.

Subsequently, the second module processes these activity-specific traces, storing them in files with .dtrace format compatible with Daikon [73], an implementation of the system for dynamic detection of likely invariants (Section 6.4.1.2). Additionally, this module generates declaration files in the .decls format for each data trace. Following this, Daikon is executed using these files to discover data conditions for each activity.

Moving forward, the third module receives as input both the data trace and the discovered data conditions of all activities linked to an artifact. Its primary function involves computing the Information Gain ($\mathcal{IG}$) of data conditions for each activity, encompassing their conjunctions. The output of this module is a table containing data conditions as headers and individual activity lines displaying the $\mathcal{IG}$ associated with their respective data conditions.

Finally, the fourth module undertakes the clustering process. Initially, it utilizes the table generated by the previous module to execute the clustering algorithm. This phase focuses on discovering of stages and nested stages. Subsequently, the module assigns guards to each stage based on the clustering outcomes.

6.5.2 Case study

For our case study, we use our motivating example (Cryptokitties) and acquired its associated ACEL log from Ethereum, using our extraction method detailed in Chapter 5.

We present a concise depiction of the kitty artifact's discovered lifecycle in Table 6.1 and visually illustrate it in Figure 6.3. This lifecycle encapsulates two atomic stages, namely $S1$ and $S2$, containing the activities Birth and Breeding, respectively. Notably, the data conditions (K.milestone == Pregnant) and (K.breedingWith.Kitty.milestone == FutureFather) associated with the Birth stage result from the closure of the Breeding stage. This is a behavioral dependency which implies that the Breeding stage consistently precedes the Birth stage. Additionally, the condition ('K.breedingWith.Kitty'.milestone) denotes an internal event, signifying a reflexive interaction exhibiting a cardinality of one, where a kitty instance maintains a 'breedingWith' relation with the primary kitty instance.

Moving to the auction artifact, Table 6.2 showcases a segment of the discovered lifecycle, visually represented in Figure 6.4. This lifecycle unfolds as a single stage ($S1$) comprising two atomic stages housing the activities CompleteAuction and CancelAuction. The data condition (A.milestone == Created) signifies the necessity for an auction to be in a 'Created' state to undergo completion or cancellation. This condition acts as a guard, preventing both completion and cancellation actions from occurring simultaneously on the same auction. Indeed, as depicted in the motivating example, once an auction is created, it can only proceed to completion or be canceled.

Table 6.1: Excerpt of a discovered kitty lifecycle.

Stages	Guards
S1 (Birth)	(K.cooldownPeriod < Timestamp and K.milestone == Pregnant and K.breedingWith.Kitty.milestone == FutureFather)
S2 (Breeding)	(K.milestone == Transferred or K.milestone == Sold or K.milestone == BecameMother)

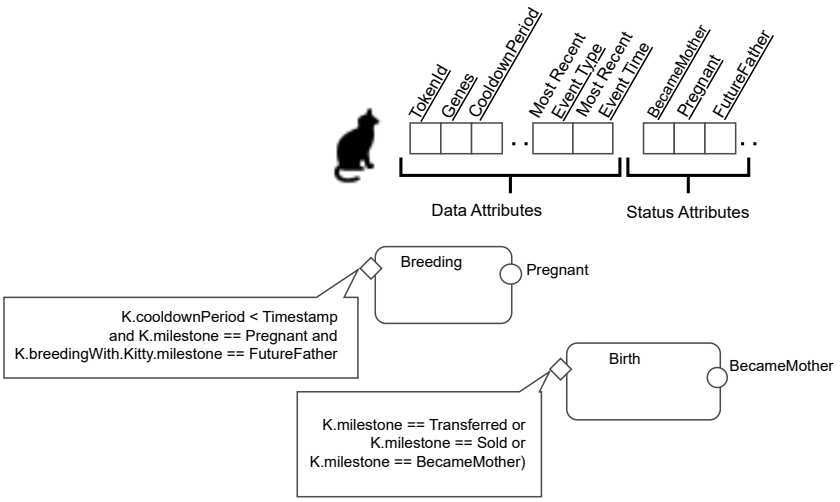

Figure 6.3: Discovered GSM model: kitty lifecycle

6.5.3 Evaluation

In this section, we assess the effectiveness of our approach concerning the discovery of data conditions, interactions, and the clustering algorithm's performance.

6.5.3.1 Evaluation of Guards and Interactions Discovery

To evaluate our approach, we use as a reference the GSM model of our motivating example Cryptokitties as derived from the application's whitepaper [60]. The Birth stage within this model encompasses two data conditions (k.'Pregnant' and K.cooldownPeriod < Timestamp)), along with one external event (k.'giveBirth'.onEvent()).

Table 6.2: Excerpt of a discovered Auction lifecycle.

Stages	Guards
S1 (CompleteAuction, CancelAuction)	(A.milestone == Created)

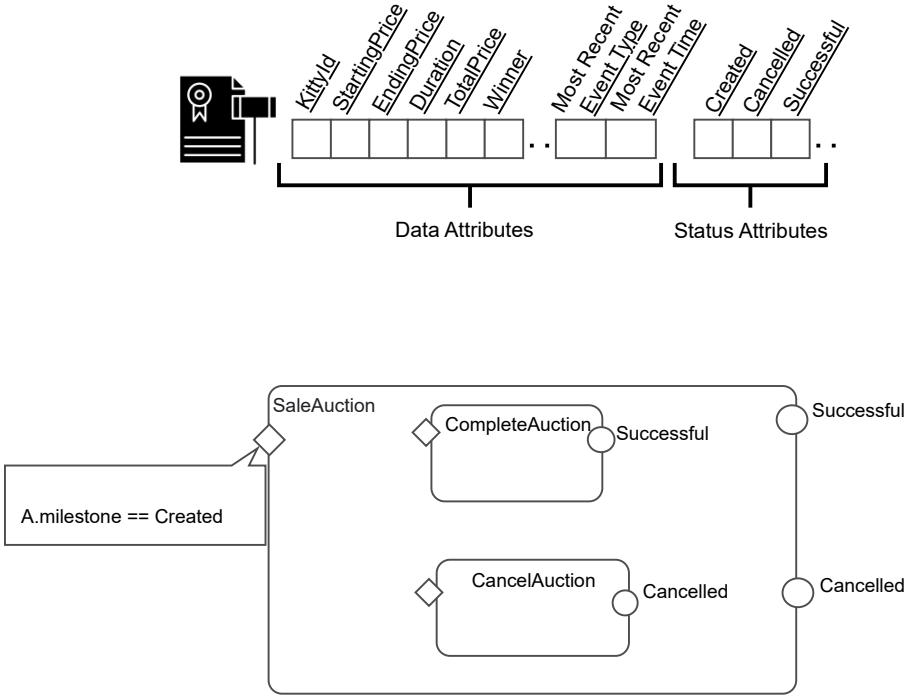

Figure 6.4: Discovered GSM model: auction lifecycle

Our approach successfully discovered these data conditions, in addition to revealing another pertinent condition (K.breedingWith.Kitty.milestone = FutureFather). The accuracy of this discovery is evident as this new condition contrasts with the existing one (not k.'siringWithId'.'FutureFather') within the Breeding stage, effectively distinguishing between the two stages.

Although external events were excluded, as explained in Section 6.3, our approach identified a reflexive interaction, embodied by the attribute K.breedingWith.Kitty.milestone, despite its absence in the visual representation of interactions in the GSM model of the motivating example. This discovery is important as it indicates the mutual influence between the lifecycles of the Father and the Mother.

However, our approach encountered a limitation in discovering the `Procreation` stage. This inability stemmed from the absence of the condition (K.cooldownPeriod < Timestamp) for the Birth activity in the extracted log. The issue arose due to the logging mechanism used in the smart contract code of Cryptokitties only registering the 'cooldown' attribute in events linked to Breeding, overlooking Birth-related events. A more precise logging mechanism could have facilitated the discovery of the `Procreation` stage.

This insight holds significance for the DApp redesign phase, and offers valuable data for conformance checking techniques. Accurate logging would have prompted DApp developers to reinforce guard conditions preceding specific activities.

Furthermore, our approach effectively discovered the SaleAuction stage, as depicted in the GSM model of the motivating example, comprising two atomic stages: CompleteAuction and CancelAuction. However, the guards for these atomic stages were not discovered, as they primarily consisted of external events that were exclueded.

6.5.3.2 Nested Stages Discovery Evaluation

We assess the effectiveness of our hierarchical clustering-based approach using the silhouette coefficient [76], a metric commonly used to evaluate clustering algorithms regarding the cohesion and separation of clusters.

Definition 6.5.1 (Silhouette Coefficient) *The silhouette coefficient for a sample point within a cluster is computed as:*

$$S = \frac{b - a}{max(a, b)}$$

Where a represents the average distance between the sample point and other points within the same cluster, and b the minimum average distance between the sample point and points in other clusters.

The silhouette coefficient ranges from -1 to 1, where a score over 0 and close to 1 indicates a well-separated and correctly clustered data, a score equal to 0 indicates overlapping clusters, while a score lower than 0 and nearer to -1 reflects incorrect clustering.

The discovered stages $S1$ and $S2$ have a silhouette coefficient of 1. Each cluster contained only one point, and they had no common conditions, resulting in a distance of 1 between them. Consequently, our algorithm successfully generated dense and well-separated clusters.

6.5.4 Discussion

In this section we discuss the performance of our discovery approach and the potential of ACEL. The accuracy of our discovered lifecycles stems from our specific definition of artifact-centric process instances (Section 6.4.1.1) and our utilization of ACEL logs. Typically, convergence and divergence issues surface when events related to artifact objects are duplicated within an artifact-centric process instance [8]. Fortunately, our approach mitigated this concern by collecting data from related events and disregarding their associated activities in the stage discovery process. However, one could argee that potential convergence problems might persist in cases of reflexive interactions. For instance, the Birth event linked to multiple instances of the kitty artifact could lead to duplications. Nevertheless, ACEL's support for transition relations, where events can impact an instance without explicitly being part of its trace, resolves such issues. This ensures that

an event associated with the Birth activity influences only the mother, and merely affects the father and the newborn kitty.

This evaluation also underscores ACEL's capability to enhance process mining outcomes for artifact-centric processes, as discussed in Chapter 5. Notably, our approach successfully derived the GSM model from ACEL logs without necessitating additional domain knowledge. Leveraging ACEL enabled the identification of a novel interaction type (reflexive interactions), which was overlooked in prior works, to the best of our knowledge. Prior research primarily focused on interactions between distinct artifact types, whereas ACEL empowered us to uncover interactions between instances of the same artifact type. The ability of ACEL to to enhance process mining is discussed below:

6.5.4.1 Depiction of Reality

ACEL's support for object evolution facilitated the discovery of lifecycles, especially guards and milestones. Each event in ACEL records the state changes of the affected artifact, ensuring adherence to reality. Since ACEL also avoids the problem of deficiency (see Section 4.2), the log are not missing any event and thus the discovered lifecycles are complete.

6.5.4.2 Convergence

The absence of convergence issues in our method is attributable to our data collection strategy, focusing on related events solely for data conditions collection while disregarding their associated activities during stage discovery. ACEL's ability to access attributes of related artifact-centric objects further supported this approach. Additionally, relying on a relation's end as the conclusion of an interaction between two artifact instances minimized the occurrence of irrelevant data conditions. Indeed, the end of relation being detectable through an attribute was made possible due to ACEL's relational evolution tracking.

6.5.4.3 Denormalization

The discovery of artifact interactions was facilitated by ACEL's comprehensive capture of the relational model. Consequently, the information model is easily reconstructed.

6.5.4.4 Transition

ACEL's inclusion of the transition concept was pivotal in averting convergence concerns in reflexive interactions. For instance, while the Birth event may be linked to multiple kitty artifact instances, ACEL's transition relations ensure that the event only affects the mother, inherently influencing the father and the newborn kitty.

6.6 Conclusion

In this chapter, we answered the question raised in the research problem section (Section 1.3), which is: How to discover artifact-centric process models from artifact-centric event logs ?. We proposed a technique to discover GSM models from ACEL logs using hierarchical clustering. Our approach used a version of hierarchical clustering where similarity is determined by common data conditions with the same information gain. We proposed for that purpose a new similarity function and criteria for cluster merging. The data conditions we use to compute the similarity matrix were discovered through an invariant detection mechanism. We also proposed a naming system for the attributes of related artifact to add semantic and better understand the nature of interactions occurring between artifacts.

We implemented and tested our approach on Cryptokitties to evaluate the performance of the discovery algorithm and the pertinence of the results. Using the silhouette metric we established that our hierarchical algorithm produced accurate results, i.e., dense and well-separated clusters. The results of the evaluation also showed that our approach discovers stages in accordance with the reality captured in the log. In particular we noted that a lack of information in the log, due to limited logging in the smart contract, prevented us from discovering one stage because the events were missing data. This data would have led to the discovery of more data conditions and consequently to the discovery of more stages. Additionally, the tests showed that our approach accurately discovers interactions between artifacts, in particular a novel type of interactions we called reflexive interactions.

Chapter 7

Conclusion and Future Work

Contents

In this chapter we summarize our contributions that provide an answer to our thesis research question : How to discover artifact-centric models from blockchain data ?. Following the summary of our work (Section 7.1), we discuss our future work (Section 7.2).

7.1 Contributions

The history of process management, from nature's intricate processes to industrial evolution, led to the emergence of Business Process Management (BPM), fostering operational optimization. The arrival of Blockchain in 2009 heralded a paradigm shift, holding potential for BPM advancement but introducing challenges in process execution, modeling complexity and analysis [34]. Traditional process modeling focused on control flows, neglecting data flow intricacies, while existing event data formats limited capturing multi-object processes, resulting in problematic and erroneous process mining outcomes. Consequently, artifact-centric process mining emerged as a solution [8], aiming to bridge this gap by exploring multi-object, interdependent processes but faces challenges in data collection, logging format compatibility, and discovery techniques alignment with artifact-centricity. The benefits of blockchain, in terms of trustworthiness of event data, also extend to artifact-centric process mining. This benefit is mutual as artifact-centric process mining on blockchain data also offers an opportunity for establishing trust in processes executed through smart contracts and understanding the business perspective of blockchain applications. The artifact-centric perspective is more pertinent, in our opinion, for blockchain

as most its applications are executed in an artifact-centric way, where the execution is driven by the state changes of business entities.

In this thesis we aimed to foster the mutual benefits between blockchain and artifact-centric process mining by solving the challenges that arise in this specific context. To that end we set five main objectives: (1) Propose an artifact-centric logging format that supports artifact-centric elements while avoiding redundancy; (2) Identify the artifact-centric event data elements present in blockchain data; (3) Propose a mapping from blockchain data to artifact-centric event data elements; (4) Propose an automate approach for the collection of artifact-centric event data from blockchain data and the generation of artifact-centric event logs from these event data; (5) Propose a technique to discover artifact-centric process models from the generated artifact-centric event logs. Consequently, we proposed three major contributions in order to fulfill these objectives.

In the first contribution, which covers the first objective, we introduced the ACEL (Artifact-Centric Event Log) format [35], presented in Chapter 4, which emerged as a pivotal milestone in addressing the shortcomings of existing logging formats. In order to propose this format we analyzed the specifications, limitations and potential of existing object-centric logging format. After this analysis, we made the decision to extend the object-centric logging standard OCEL [17] for more optimisation and applicability (Section 4.2). To obtain ACEL, we enhanced OCEL with novel concepts that allowed the capturing of object evolution as well as relations between objects and their evolution. The novel model we proposed focused on avoiding redundancy and optimizing storage space, while allowing easier processing of logs for process mining purposes. Through a qualitative evaluation of the potential of ACEL to improve artifact-centric process mining results in comparison with other logging formats, we showed its clear ability to store all the element of artifact-centric event data in a richer, more efficient, and accurate manner. In particular, we showed that it captures all the information present in artifact-centric event data without any redundancy in data storage.

In the second contribution, presented in Chapter 5, we fulfilled the second, third and fourth objectives by proposing an approach which covers the steps required to automate the collection of artifact-centric event data elements from blockchain data and the generation of ACEL logs from this data. To streamline this process, we first defined a template, for an easier creation of a configuration file, that informs the user of the correct structure of the configuration file. Then, we proposed an approach, which first automates the extraction of smart contract events from the blockchain and then automates the process of converting the smart contract events into ACEL elements to generate an ACEL log. Additionally, we proposed an approach to apply existing process mining techniques on ACEL, even though they are not inherently compatible. We proposed an ACEL filtering technique to obtain an OCEL log, while preserving the evolution of objects. We implemented our extraction and generation of ACEL logs approach as an online tool. We evaluated this approach through a case study on Cryptokitties, which showcased its feasibility and

further demonstrated ACEL's performance compared to other logging formats and highlighted its strengths. We then evaluated our approach for applying existing process mining techniques on ACEL using logs from Cryptokitties and Augur. We extracted ACEL logs for these two applications using our extraction and generation approach and then applied our filtering technique to obtain ACEL logs structured according to the OCEL standard. We then discovered OC-DFGs using an existing process mining technique applied to the filtered ACEL logs. Through this experiment, we demonstrated that with proper filtering, ACEL logs are compatible with current process mining techniques, enabling the discovery of artifact-centric models. Our proposed filtering method allowed us to generate OCEL logs from ACEL while preserving object evolution. The evaluation results solidified ACEL's advantages over OCEL in discovering artifact-centric models and revealed pertinent insights within the generated models. It also further motivated the needs for novel process mining techniques compatible with ACEL to reap the full potential of this novel logging format.

In the third contribution, presented in Chapter 6, we fully achieved the fifth objective, through our novel discovery approach which takes as input ACEL logs and discovers GSM models [36]. We divided the discovery of GSM models into two key steps: (i) Identifying the information model (comprising artifact attributes and relationships) and (ii) Discovering the lifecycle of each artifact (including stages, their guards, milestones, and hierarchical structure). We argued that the information model was already present in ACEL and thus focused on the second step. We demonstrated that we can discover lifecycles and interactions by discovering data conditions and nested stages. To do so, we proposed an approach which encompasses two techniques: the first discovers data conditions; and the second utilizes these data conditions to discover nested stages. Our data condition discovery technique is based on a existing invariants detection mechanism used in the literature to discover branching conditions [74]. We adapted this mechanism to our context and proposed a new definition for artifact-centric process instances and activity data traces. This new definition allowed us to avoid convergence and divergence while allowing a more efficient discovery of data conditions. Additionally, we proposed a naming convention for related artifact attributes to enhance semantic understanding of artifact interactions. Our discovery technique for nested stages, is based on hierachical clustering algorithms. We adapted the hierarchical clustering approach to rely on common data conditions to determine similarity. In our proposed clustering, we defined a new similarity function and a new linkage criterion. The similarity function computes the closeness of two activities of stages based on the information gain of their common data conditions. The linkage criterion we defined dictates that all activities within a cluster must have the same similarity score with all activities of another cluster before they can be merged. We implemented our approach in a tool and evaluated its efficiency using the motivating example. Through the silhouette metric, we verified that our hierarchical clustering algorithm generated precise results, showcasing well-defined and distinct clusters. Our evaluation also confirmed the

accurate discovery of stages consistent with the log's captured reality. Furthermore, our tests highlighted the successful identification of interactions between artifacts, including a novel type called reflexive interactions. The results of this evaluation finishes to prove that ACEL's ability to efficiently store artifact-centric event data marks a significant advancement for artifact-centric process mining.

Hence, all three of our contributions allowed us to answer the research question this thesis was based on. Through all the implementations and evaluations we were able to achieve the objectives that were set in the beginning of this thesis (Section 1.4).

7.2 Future work

Our work pioneers the application of process mining to artifact-centric processes within the blockchain context, opening new avenues for understanding and optimizing complex workflows with many perspectives. Hence, it opens several research directions. We will focus in the short term on improving our two approaches (Chapters 5 and 6) and their performance as well as addressing, in the long term, other related research issues such how to benefit from the event data of IoT-aware processes [77] for artifact-centric process mining. In Section 7.2.1 we present our future work to enhance our extraction approach, then we present in Section 7.2.2 an overview of the limitations of the GSM discovery approach we wish to address in future work and the features we plan to add to it.

7.2.1 Fully automating ACEL logs extraction

Our ACEL logs extraction approach, in Chapter 5, relies on domain knowledge to obtain accurate ACEL logs from blockchain applications. However, the domain knowledge collection (configuration phase), although supported through a template and an online tool, lacks more automation. We plan to fully automate this part through text mining techniques to enhance the configuration file creation and expedite this process, reducing the need for manual intervention. This approach involves utilizing the user-provided smart contract (SC) address to retrieve its corresponding source code. Upon accessing this code, our proposed strategy involves an initial phase to identify "Structs" [1], serving as essential data structures utilized by SC developers for data aggregation. These identified Structs are then presented as prospective artifacts for consideration by domain experts. The experts possess the flexibility to either adopt these proposed Structs as ready-to-use artifacts or modify them to create novel artifacts.

Furthermore, our strategy involves detecting SC events within the source code, as the entry points for mapping rules. Thus, for each smart contract event found in the source code, we will guide the user in the creation of a mapping rule from the data of the event to

[1] `https://docs.soliditylang.org/en/latest/types.html`

ACEL elements. By analyzing the parameters associated with each smart contract event, we can suggest the artifacts they influence. This proactive suggestion aids domain experts in formulating event mapping rules, contributing to automating the configuration process.

Lastly, to suggest relations between artifacts, we will focus on artifacts that are often associated with the same smart contract events. These suggestions will enable domain experts to validate and define the attributes of these suggested relations as per their expertise and requirements. This comprehensive strategy anticipates a significant reduction in the manual effort required for the creation of configuration files while ensuring an expert-driven refinement process.

7.2.2 Optimizing the discovery algorithm

Regarding our GSM discovery approach, we are working on improving its two techniques (data conditions discovery and stages and nested stages discovery). The data condition discovery technique relies on Daikon for invariants detection, this latter uses declaration files to indicate pertinent variable to compare through a comparability parameter. We have considered so far that this parameter was user provided. However, we can discover this parameter and help guide the user in decision making. Comparability is basically how correlated two variables are, and that information can be derived from the ACEL logs. Indeed, from an ACEL log, we can deduce correlations between variables, for instance, by identifying variables frequently appearing together in the ObjectChanges list. For example, if the variables a et b of an artifact often change values, i.e., appear together in the ObjectChanges list, this can suggest that they are correlated and thus comparable.

The second part of our future work entails addressing the limitations which arise from the hierarchical clustering technique, particularly its incapacity to discover stage-specific milestones and constraints limiting a single guard per stage. Future work entails addressing these shortcomings by integrating post-conditions and disjunctive data conditions and refining clustering methodologies. By incorporating post-conditions and disjunctive data conditions, we aim to enhance nested stage discovery and enable multiple guards per stage. Refining clustering methodologies by implementing thresholds for Information Gain will fortify accuracy and applicability.
We also aim to broaden our evaluation scope to include diverse and complex processes and test our contributions on data sources other than blockchain. For example, we aim to adapt our extraction algorithm to collect event data from process-aware information systems and generate ACEL logs. This will allow us to cover more diverse process than the use cases found in blockchain, in order to show the potential of ACEL as well as that of our artifact-centric discovery approach.

Another perspective of our work, is to apply our approach to a context of IoT-aware artifact-centric process. This includes processes where IoT devices act as resources for

the execution of activities and those where they are assimilated to artifacts. Considering this new context will raise interesting challenges, such as how to handle the continuous flow of information coming from the devices. Furthermore, we are inclined to explore the execution of IoT-aware artifact-centric processes on blockchain and then apply our discovery approach on their blockchain event data.

Bibliography

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2013.

[2] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*, 2nd ed. Springer, 2018.

[3] T. Ahmad and A. V. Looy, "Reviewing the historical link between business process management and IT: making the case towards digital innovation," in *13th International Conference on Research Challenges in Information Science, RCIS 2019, Brussels, Belgium, May 29-31, 2019*. IEEE, 2019, pp. 1–12.

[4] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data, BigData Congress 2017, Honolulu, HI, USA, June 25-30, 2017*. IEEE Computer Society, 2017, pp. 557–564.

[5] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[6] J. Kunchala, J. Yu, and S. Yongchareon, "A survey on approaches to modeling artifact-centric business processes," in *WISE Workshops - 15th International Workshops IWCSN, Org2, PCS, and QUAT, Thessaloniki, Greece, October 12-14*, ser. LNCS, vol. 9051. Springer, 2014, pp. 117–132.

[7] R. Hull, "Artifact-centric business process models: Brief survey of research results and challenges," in *OTM, Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE, Monterrey, Mexico, November 9-14*, ser. LNCS, vol. 5332. Springer, 2008, pp. 1152–1163.

[8] D. Fahland, "Artifact-centric process mining," in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya, Eds. Springer, 2019.

[9] M. Dumas, W. M. Van der Aalst, and A. H. Ter Hofstede, *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005.

[10] J. F. Chang, *Business process management systems: strategy and implementation.* CRC Press, 2016.

[11] W. M. Van der Aalst, "Extracting event data from databases to unleash process mining," in *BPM-Driving innovation in a digital world.* Springer, 2015, pp. 105–128.

[12] M. Chinosi and A. Trombetta, "BPMN: an introduction to the standard," *Comput. Stand. Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[13] "IEEE standard for extensible event stream (XES) for achieving interoperability in event logs and event streams," *IEEE Std 1849-2016*, pp. 1–50, 2016.

[14] A. Berti and W. M. P. van der Aalst, "OC-PM: analyzing object-centric event logs and process models," *Int. J. Softw. Tools Technol. Transf.*, vol. 25, no. 1, pp. 1–17, 2023.

[15] G. Li and R. M. de Carvalho, "Dealing with artifact-centric systems: a process mining approach," in *Proceedings of the 9th International Workshop on Enterprise Modeling and Information Systems Architectures, Rostock, Germany, May 24th - 25th*, ser. CEUR Workshop Proceedings, vol. 2097. CEUR-WS.org, 2018, pp. 80–84.

[16] G. Li, E. G. L. de Murillas, R. M. de Carvalho, and W. M. P. van der Aalst, "Extracting object-centric event logs to support process mining on databases," in *Information Systems in the Big Data Era - CAiSE Forum, Tallinn, Estonia, June 11-15*, ser. LNBIP, vol. 317. Springer, 2018, pp. 182–199.

[17] A. F. Ghahfarokhi, G. Park, A. Berti, and W. M. P. van der Aalst, "OCEL: A standard for object-centric event logs," in *New Trends in ADBIS, Tartu, Estonia*, ser. CCIS, vol. 1450, 2021, pp. 169–175.

[18] L. Moctar-M'Baba, M. Sellami, W. Gaaloul, and M. F. Nanne, "Blockchain logging for process mining: a systematic review," in *55th Hawaii International Conference on System Sciences, HICSS, Virtual Event / Maui, Hawaii, USA, January 4-7.* ScholarSpace, 2022, pp. 1–10.

[19] C. Klinkmüller, A. Ponomarev, A. B. Tran, I. Weber, and W. van der Aalst, "Mining blockchain processes: extracting process mining data from blockchain applications," in *BPM Blockchain and CEE Forum, Vienna, Austria*, ser. LNBIP, vol. 361. Springer, 2019, pp. 71–86.

[20] R. Mühlberger, S. Bachhofner, C. D. Ciccio, L. García-Bañuelos, and O. López-Pintado, "Extracting event logs for process mining from data stored on the blockchain," in *BPM International Workshops, Vienna, Austria, September 1-6*, ser. LNBIP, vol. 362. Springer, 2019, pp. 690–703.

[21] N. Y. Wirawan *et al.*, "Incorporating Transaction Lifecycle Information in Blockchain Process Discovery," in *Blockchain Technology for IoT Applications.* Singapore: Springer Singapore, 2021, pp. 155–172.

[22] V. Popova and M. Dumas, "From petri nets to guard-stage-milestone models," in *BPM Workshops, Tallinn, Estonia, September 3*, ser. LNBIP, vol. 132, 2012, pp. 340–351.

[23] V. Popova, D. Fahland, and M. Dumas, "Artifact lifecycle discovery," *Int. J. Cooperative Inf. Syst.*, vol. 24, no. 1, pp. 1 550 001:1–1 550 001:44, 2015.

[24] W. M. P. van der Aalst, "Object-centric process mining: Dealing with divergence and convergence in event data," in *17th International Conference, SEFM, Oslo, Norway, September 18-20*, ser. LNCS, vol. 11724. Springer, 2019, pp. 3–25.

[25] R. Hull, E. Damaggio, F. Fournier, M. Gupta *et al.*, "Introducing the guard-stage-milestone approach for specifying business entity lifecycles," in *WS-FM - 7th Workshop, Hoboken, NJ, USA, September 16-17.*, ser. LNCS, vol. 6551, 2010, pp. 1–24.

[26] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[27] W. van der Aalst, "Process mining: Overview and opportunities," *ACM Trans. Manage. Inf. Syst.*, vol. 3, no. 2, pp. 7:1–7:17, place: New York, NY, USA Publisher: Association for Computing Machinery.

[28] W. M. van der Aalst, "Process discovery: Capturing the invisible," *IEEE Computational Intelligence Magazine*, vol. 5, no. 1, pp. 28–41, 2010.

[29] W. van der Aalst and W. van der Aalst, "Conformance checking," *Process Mining: Data Science in Action*, pp. 243–274, 2016.

[30] W. M. Van der Aalst and B. F. van Dongen, "Discovering workflow performance models from timed logs," in *International Conference on Engineering and Employment of Cooperative Information Systems.* Springer, 2002, pp. 45–63.

[31] A. Bolt, W. M. van der Aalst, and M. De Leoni, "Finding process variants in event logs: (short paper)," in *On the Move to Meaningful Internet Systems. OTM Conferences: Confederated International Conferences: CoopIS, C&TC, and ODBASE, Rhodes, Greece, October 23-27.* Springer, 2017, pp. 45–52.

[32] L. S. Sankar, M. Sindhu, and M. Sethumadhavan, "Survey of consensus protocols on blockchain applications," in *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2017.

[33] S. N. Khan, F. Loukil, C. G. Guegan, E. Benkhelifa, and A. Bani-Hani, "Blockchain smart contracts: Applications, challenges, and future trends," *Peer-to-Peer Netw. Appl.*, vol. 14, no. 5, pp. 2901–2925, 2021.

[34] J. Mendling, I. Weber, W. M. P. van der Aalst, J. vom Brocke, C. Cabanillas, F. Daniel *et al.*, "Blockchains for business process management - challenges and opportunities," *ACM Trans. Manag. Inf. Syst.*, vol. 9, no. 1, pp. 4:1–4:16, 2018.

[35] L. M. M'Baba, N. Assy, M. Sellami, W. Gaaloul, and M. F. Nanne, "Extracting artifact-centric event logs from blockchain applications," in *IEEE International Conference on Services Computing, SCC, Barcelona, Spain, July 10-16*. IEEE, 2022, pp. 274–283.

[36] L. Moctar-M'Baba, M. Sellami, N. Assy, W. Gaaloul, and M. F. Nanne, "Discovering guard stage milestone models through hierarchical clustering," in *CoopIS , Groningen, The Netherlands, October 30 - November 3, Proceedings*, ser. Lecture Notes in Computer Science, vol. 14353. Springer, 2023, pp. 239–256.

[37] L. Moctar-M'Baba, N. Assy, M. Sellami, W. Gaaloul, and M. F. Nanne, "Process mining for artifact-centric blockchain applications," *Simul. Model. Pract. Theory*, vol. 127, p. 102779, 2023.

[38] A. Goossens, J. D. Smedt, J. Vanthienen, and W. M. P. van der Aalst, "Enhancing data-awareness of object-centric event logs," in *Process Mining Workshops - ICPM 2022 International Workshops, Bozen-Bolzano, Italy, October 23-28*, ser. Lecture Notes in Business Information Processing, vol. 468. Springer, 2022, pp. 18–30.

[39] B. Ekici *et al.*, "Data Cleaning for Process Mining with Smart Contract," in *2019 4th International Conference on Computer Science and Engineering (UBMK)*, Sep. 2019, pp. 1–6.

[40] E. Brinckman *et al.*, "Techniques and applications for crawling, ingesting and analyzing blockchain data," in *2019 International Conference on Information and Communication Technology Convergence, ICTC 2019, Jeju Island, Korea (South)*. IEEE, Oct 2019, pp. 717–722.

[41] D. Zimina and D. Mouromtsev, "Applying blockchain technology for improvement of the educational process in terms of data processing," in *11th Majorov International Conference on Software Engineering and Computer Systems (MICSECS 2019), Saint Petersburg*, ser. CEUR Workshop Proceedings, vol. 2590. CEUR-WS.org, Dec 2019.

[42] S. van Engelenburg *et al.*, "Design of a software architecture supporting business-to-government information sharing to improve public safety and security - combining

business rules, events and blockchain technology," *J. Intell. Inf. Syst.*, vol. 52, no. 3, pp. 595–618, 2019.

[43] S. Tönnissen and F. Teuteberg, "Using blockchain technology for cross-organizational process mining - concept and case study," in *Business Information Systems - 22nd International Conference, BIS 2019, Seville, Spain*, ser. Lecture Notes in Business Information Processing, vol. 354. Springer, June 2019, pp. 121–131.

[44] C. D. Ciccio *et al.*, "Blockchain-based traceability of inter-organisational business processes," in *Business Modeling and Software Design - 8th International Symposium, BMSD 2018, Vienna, Austria*, ser. Lecture Notes in Business Information Processing, vol. 319. Springer, July 2018, pp. 56–68.

[45] O. López-Pintado *et al.*, "CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain," *CoRR*, vol. abs/1808.03517, 2018.

[46] R. Mühlberger *et al.*, "Extracting event logs for process mining from data stored on the blockchain," in *Business Process Management Workshops - BPM 2019 International Workshops, Vienna, Austria*, ser. Lecture Notes in Business Information Processing, vol. 362. Springer, Sept 2019, pp. 690–703.

[47] F. Duchmann and A. Koschmider, "Validation of smart contracts using process mining," in *11th Central European Workshop on Services and their Composition, Bayreuth, Germany*, ser. CEUR Workshop Proceedings, vol. 2339. CEUR-WS.org, Feb 2019, pp. 13–16.

[48] C. W. Günther *et al.*, "Disco: Discover your processes," in *Demonstration Track of the 10th International Conference*, ser. CEUR Workshop Proceedings, vol. 940. CEUR-WS.org, 2012, pp. 40–44.

[49] B. F. van Dongen *et al.*, "The prom framework: A new era in process mining tool support," in *Applications and Theory of Petri Nets, 26th International Conference, ICATPN, June, Miami, USA*, ser. Lecture Notes in Computer Science, vol. 3536. Springer, 2005, pp. 444–454.

[50] F. Corradini *et al.*, "Enabling auditing of smart contracts through process mining," in *From Software Engineering to Formal Methods and Tools, and Back*, ser. Lecture Notes in Computer Science, vol. 11865. Springer, 2019, pp. 467–480.

[51] C. Klinkmüller *et al.*, "Efficient logging for blockchain applications," *CoRR*, vol. abs/2001.10281, 2020.

[52] M. Müller and P. Ruppel, "Process mining for decentralized applications," in *IEEE International Conference on Decentralized Applications and Infrastructures, DAPPCON 2019, Newark, CA, USA*. IEEE, Apr 2019, pp. 164–169.

[53] A. Berti and W. M. P. van der Aalst, "Extracting multiple viewpoint models from relational databases," *CoRR*, vol. abs/2001.02562, 2020.

[54] E. H. J. Nooijen, B. F. van Dongen, and D. Fahland, "Automatic discovery of data-centric and artifact-centric processes," in *Business Process Management Workshops - BPM International Workshops, Tallinn, Estonia, September 3. Revised Papers*, ser. Lecture Notes in Business Information Processing, vol. 132. Springer, 2012, pp. 316–327.

[55] D. Fahland, "Process mining over multiple behavioral dimensions with event knowledge graphs," in *Process Mining Handbook*, ser. Lecture Notes in Business Information Processing, W. M. P. van der Aalst and J. Carmona, Eds. Springer, 2022, vol. 448, pp. 274–319.

[56] X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland, "Discovering interacting artifacts from ERP systems," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, pp. 861–873, 2015.

[57] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst, "Composite state machine miner: Discovering and exploring multi-perspective processes," in *Proceedings of the BPM Demo Track Co-located with the 14th International Conference on Business Process Management (BPM), Rio de Janeiro, Brazil, September 21,*, ser. CEUR Workshop Proceedings, L. Azevedo and C. Cabanillas, Eds., vol. 1789. CEUR-WS.org, 2016, pp. 73–77.

[58] V. Popova and M. Dumas, "Discovering unbounded synchronization conditions in artifact-centric process models," in *BPM Workshops, Beijing, China, August 26*, ser. LNBIP, vol. 171, 2013, pp. 28–40.

[59] H. Nguyen, M. Dumas, A. H. M. ter Hofstede, M. L. Rosa *et al.*, "Stage-based discovery of business process models from event logs," *Inf. Syst.*, vol. 84, pp. 214–237, 2019.

[60] Cryptokitties, "Cryptokitties: Collectible and breedable cats empowered by blockchain technology. white pa-purr," 2018.

[61] N. Lohmann and M. Nyolt, "Artifact-centric modeling using BPMN," in *Service-Oriented Computing - ICSOC Workshops - International Workshops WESOA, NFPSLAM-SOC, and Satellite Events, Paphos, Cyprus, December 5-8. Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 7221. Springer, 2011, pp. 54–65.

[62] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2014. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[63] M. Alharby and A. van Moorsel, "Blockchain-based smart contracts: A systematic mapping study," *CoRR*, vol. abs/1710.06372, 2017.

[64] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, and H. Lee, "Systematic review of security vulnerabilities in ethereum blockchain smart contract," *IEEE Access*, vol. 10, pp. 6605–6621, 2022.

[65] A. Berti and W. M. P. van der Aalst, "OC-PM: analyzing object-centric event logs and process models," *Int. J. Softw. Tools Technol. Transf.*, vol. 25, no. 1, pp. 1–17, 2023.

[66] W. M. P. van der Aalst, "A practitioner's guide to process mining: Limitations of the directly-follows graph," in *CENTERIS / ProjMAN / HCist, Sousse, Tunisia*, ser. Procedia Computer Science, vol. 164. Elsevier, 2019, pp. 321–328.

[67] J. H. Perkins and M. D. Ernst, "Efficient incremental algorithms for dynamic detection of likely invariants," in *Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2004, Newport Beach, CA, USA, October 31 - November 6*. ACM, 2004, pp. 23–32.

[68] T. Mitchell, "Decision tree learning," *Machine learning*, vol. 414, pp. 52–78, 1997.

[69] S. Suthaharan, *Decision Tree Learning*. Boston, MA: Springer US, 2016, pp. 237–269.

[70] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Prentice-Hall, Inc., 1988.

[71] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., 1990.

[72] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. Gupta *et al.*, "Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events," in *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems, DEBS, New York, NY, USA, July 11-15*. ACM, 2011, pp. 51–62.

[73] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco *et al.*, "The daikon system for dynamic detection of likely invariants," *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 35–45, 2007.

[74] M. de Leoni, M. Dumas, and L. García-Bañuelos, "Discovering branching conditions from business process execution logs," in *Fundamental Approaches to Software Engineering - 16th International Conference, FASE, Rome, Italy, March 16-24*, ser. Lecture Notes in Computer Science, vol. 7793. Springer, 2013, pp. 114–129.

[75] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *The computer journal*, vol. 26, pp. 354–359, 1983.

[76] J. Palacio-Niño and F. Berzal, "Evaluation metrics for unsupervised learning algorithms," *CoRR*, vol. abs/1905.05667, 2019.

[77] Z. Maamar, E. Kajan, I. Guidara, L. Moctar-M'Baba, and M. Sellami, "Bridging the gap between business processes and iot," in *IDEAS, Seoul, Republic of Korea, August 12-14*. ACM, 2020, pp. 2:1–2:10.

Chapter 8

Appendices

8.1 Event log of Cryptokitties

The listings 8.1, 8.2, 8.3 and 8.4 show respectively the XES, XOC, OCEL and ACEL logs we manually created according to the structure of each logging format for the Cryptokitties blockchain application.

Listing 8.1: XES Log Snippet

```
<log xes.version="1.0" xes.features="nested-attributes"
openxes.version="1.0RC7">
<extension name="Time" prefix="time"
uri="http://www.xes-standard.org/time.xesext"/>
<extension name="Concept" prefix="concept"
uri="http://www.xes-standard.org/concept.xesext"/>
<global scope="event">
<date key="time:timestamp" value="1970-01-01T01:00:00+01:00"/>
<string key="concept:name" value="No global value for
concept:name defined"/>
</global>
<global scope="event">
                <string key="concept:name" value="name"/>
                <string key="org:resource" value="resource"/>
                <date key="time:timestamp"
        value="2011-04-13T14:02:31.199+02:00"/>
                <string key="Activity" value="string"/>
                <string key="Resource" value="string"/>
                <string key="genes" value="string"/>
        <string key="owner" value="string"/>
        <string key="cooldownPeriod" value="string"/>
```

```
22 </global>
23 <classifier name="Activity" keys="Activity"/>
24 <classifier name="activity classifier" keys="Activity"/>
25 <trace>
26 <string key="ident:piid" value="1240424"/>
27 <event>
28     <int key="logIndex" value="1"/>
29     <string key="concept:name" value="Conceive as Matron"/>
30     <string key="Activity" value="Breeding"/>
31     <string key="Resource" value="0xf12A13.."/>
32     <string key="sireId" value="1475706"/>
33     <string key="cooldownPeriod" value="11225643"/>
34     <date key="time:timestamp"
35     value="2023-10-23T06:11:51"/>
36 </event>
37 <event>
38     <int key="logIndex" value="2"/>
39     <string key="concept:name" value="Conceive as Sire"/>
40     <string key="Activity" value="Breeding"/>
41     <string key="Resource" value="0xf12A13.."/>
42     <string key="matronId" value="1240424"/>
43     <string key="cooldownPeriod" value="11225643"/>
44     <date key="time:timestamp" value="2023-10-23T06:11:51"/>
45 </event>
46 <event>
47     <int key="logIndex" value="3"/>
48     <string key="concept:name" value="Give Birth as Matron"/>
49     <string key="Activity" value="Birth"/>
50     <string key="Resource" value="0xf12A13.."/>
51     <string key="kittyId" value="1576916"/>
52     <string key="sireId" value="1475706"/>
53     <string key="cooldownPeriod" value="0"/>
54     <date key="time:timestamp" value="2023-10-24T10:12:36"/>
55 </event>
56 <event>
57     <int key="logIndex" value="4"/>
58     <string key="concept:name" value="Give Birth as Sire"/>
59     <string key="Activity" value="Birth"/>
60     <string key="Resource" value="0xf12A13.."/>
61     <string key="kittyId" value="1576916"/>
```

```
62      <string key="matronId" value="1240424"/>
63      <string key="cooldownPeriod" value="0"/>
64      <date key="time:timestamp" value="2023-10-24T10:12:36"/>
65  </event>
66  <event>
67      <int key="logIndex" value="5"/>
68      <string key="concept:name" value="Is Born"/>
69      <string key="Activity" value="Birth"/>
70      <string key="Resource" value="0xf12A13.."/>
71      <string key="matronId" value="1240424"/>
72      <string key="sireId" value="1475706"/>
73      <string key="genes" value="8658320..."/>
74      <string key="owner" value="0xf12A13.."/>
75      <date key="time:timestamp" value="2023-10-24T10:12:36"/>
76  </event>
77  </trace>
78  </log>
```

Listing 8.2: XOC Log Snippet

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <log xoc.version="1.0">
3  <string key="format" value="total"/>
4  <event>
5          <string key="id" value="e1"/>
6          <string key="activity" value="Breeding"/>
7          <string key="timestamp" value="23/10/2023 10:33:37"/>
8          <model>
9          <objects>
10         <object>
11                 <string key="id" value="o1"/>
12                 <string key="class" value="cat"/>
13                 <string key="tokenid" value="1240424"/>
14         <string key="genes" value="6789293"/>
15         </object>
16         <object>
17                 <string key="id" value="o2"/>
18                 <string key="class" value="cat"/>
19                 <string key="tokenid" value="1475706"/>
20                 <string key="genes" value="75607913"/>
21         </object>
```

```
22          </objects>
23          <relations>
24          <relation>
25                  <string key="id" value="e1-r1-o1"/>
26                  <string key="relation" value="r1"/>
27                  <string key="sourceobjectid" value="o1"/>
28                  <string key="targetobjectid" value="o2"/>
29          </relation>
30      </relations>
31          </model>
32          <references>
33                  <object>
34                          <string key="id" value="o1"/>
35                  </object>
36                  <object>
37                        <string key="id" value="o2"/>
38                  </object>
39          </references>
40  </event>
41  <event>
42          <string key="id" value="e2"/>
43          <string key="activity" value="breeding"/>
44          <string key="timestamp" value="23/10/2023 10:33:37"/>
45          <model>
46          <objects>
47          <object>
48                  <string key="id" value="o1"/>
49                  <string key="class" value="cat"/>
50                  <string key="tokenid" value="1240424"/>
51                  <string key="genes" value="6789293"/>
52          </object>
53          <object>
54                  <string key="id" value="o2"/>
55                  <string key="class" value="cat"/>
56                  <string key="tokenid" value="1475706"/>
57                  <string key="genes" value="75607913"/>
58          </object>
59      <object>
60                  <string key="id" value="o3"/>
61                  <string key="class" value="cat"/>
```

```
62                  <string key="tokenid" value="1576916"/>
63                  <string key="genes" value="8658320"/>
64          </object>
65          </objects>
66          <relations>
67          <relation>
68                  <string key="id" value="e1-r1-o1"/>
69                  <string key="relation" value="r1"/>
70                  <string key="sourceobjectid" value="o1"/>
71                  <string key="targetobjectid" value="o2"/>
72          </relation>
73      <relation>
74                  <string key="id" value="e2-r2-o3"/>
75                  <string key="relation" value="r2"/>
76                  <string key="sourceobjectid" value="o3"/>
77                  <string key="targetobjectid" value="o1"/>
78          </relation>
79      <relation>
80          <string key="id" value="e2-r3-o3"/>
81          <string key="relation" value="r3"/>
82          <string key="sourceobjectid" value="o3"/>
83          <string key="targetobjectid" value="o2"/>
84      </relation>
85          </relations>
86          </model>
87          <references>
88                  <object>
89                          <string key="id" value="o1"/>
90                  </object>
91                  <object>
92                          <string key="id" value="o2"/>
93                  </object>
94                  <object>
95                          <string key="id" value="o3"/>
96                  </object>
97          </references>
98  </event>
99  </log>
```

Listing 8.3: OCEL Log Snippet

```
{"ocel:global-event":{"ocel:activity":"--INVALID--"},
"ocel:global-object":{"ocel:type":"--INVALID--"},
"ocel:global-log":{"ocel:attribute-names":
["genesSequence","cooldownPeriod","mother","father",
"owner","generation","kittyId","startingPrice",
"endingPrice","duration","totalPrice","winner","composedId"],
"ocel:object-types":["cat","SaleAuction","SireAuction"],
"ocel:relation-types":["hasMother","hasFather","breedingWith",
"hasSaleAuction","hasSireAuction"],
"ocel:version":"1.0","acel:ordering":"timestamp"},

"ocel:events":{

"1":{"ocel:activity":"Breeding as Matron",
"ocel:timestamp":1511415679,
"ocel:vmap":{"resource":
"0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
"ocel:omap":["o1"]},

"2":{"ocel:activity":"Breeding as Sire",
"ocel:timestamp":1511415679,
"ocel:vmap":{"resource":
"0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
"ocel:omap":["o2"]},

"3":{"ocel:activity":"Birth as Matron",
"ocel:timestamp":1511415679,
"ocel:vmap":{"resource":
"0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
"ocel:omap":["o3"]},

"4":{"ocel:activity":"Birth as Sire",
"ocel:timestamp":1511415679,
"ocel:vmap":{"resource":
"0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
"ocel:omap":["o4"]},

"5":{"ocel:activity":"Birth as kitty",
"ocel:timestamp":1511415679,
"ocel:vmap":{"resource":
```

```
41 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
42 "ocel:omap":["o5"]}
43 }},
44 "ocel:objects":{
45
46 "o1":{"ocel:type":"cat","ocel:ovmap":
47 {"genesSequence":
48 "626837621154801616088980922659877168609154386318304496
49 6692374110716999053"}},
50
51 "o2":{"ocel:type":"cat","ocel:ovmap":
52 {"genesSequence":
53 "623332824742417442073801652020554010523726975553
54 705023219600667807529387"}},
55
56 "o5":{"ocel:type":"cat","ocel:ovmap":{"genesSequence":
57 "516352335416235417056702290154738622491807922722
58 246569050824890165376967 5"}}}}
```

Listing 8.4: ACEL Log Snippet

```
1 {"acel:global-event":{"acel:activity":"__INVALID__"},
2 "acel:global-object":{"acel:type":"__INVALID__"},
3 "acel:global-log":{"acel:attribute-names":
4 ["genesSequence","cooldownPeriod","mother","father","owner",
5 "generation","kittyId","startingPrice","endingPrice",
6 "duration","totalPrice","winner","composedId"],
7 "acel:object-types":["cat","SaleAuction","SireAuction"],
8 "acel:relation-types":
9 ["hasMother","hasFather","breedingWith","hasSaleAuction",
10 "hasSireAuction"],
11 "acel:version":"1.0","acel:ordering":"timestamp"},
12
13 "acel:events":{
14
15 "1":{"acel:activity":"Breeding",
16 "acel:timestamp":1511415679, "acel:vmap":{"resource":
17 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
18 "acel:omap":["1240424"],
19 "acel:rmap":["r1","r2","r3"],"acel:ocmap":{"1240424":
20 {"CooldownPeriod":"11225643","lifecycle":"Pregnant"},
```

```
21 "1475706":{"CooldownPeriod":"11225643",
22 "lifecycle":"FutureFather"}},
23 "acel:rcmap":{"r1":{"target":"0",
24 "changeStatus":"deletedTarget"},"r2":
25 {"target":"0","changeStatus":"deletedTarget"},"r3":
26 {"target":"0","changeStatus":"deletedTarget"}}},
27
28 "2":{"acel:activity":"Birth",
29 "acel:timestamp":1511415679, "acel:vmap":{"resource":
30 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936"},
31 "acel:omap":["1576916"],"acel:rmap":["r1","r2","r3"],
32 "acel:ocmap":{"1576916":{"owner":
33 "0x672eC49F7f7EaC25C3A2E651F67F579BB5Da8936",
34 "lifecycle":"Born"}, "1240424":{"lifecycle":"BecameMother"},
35 "1475706":{"lifecycle":"BecameFather"}},
36 "acel:rcmap":{"r1":{"target":"1475706",
37 "changeStatus":"deletedTarget"},
38 "r2":{"target":"1240424","changeStatus":"addeddTarget"},
39 "r3":{"target":"1475706","changeStatus":"addedTarget"}}}},
40
41 "acel:objects":{
42
43 "1240424":{"acel:type":"cat",
44 "acel:ovmap":{"genesSequence":
45 "626837621154801616088980922659871686091543863 1
46 830449669237411071699905 3"}},
47
48 "1475706":{"acel:type":"cat",
49 "acel:ovmap":{"genesSequence":
50 "623332824742417442073801652020554010523726975553
51 705023219600667807529387"}},
52
53 "1475706":{"acel:type":"cat",
54 "acel:ovmap":{"genesSequence":
55 "516352335416235417056702290154738622491807922
56 722465690508248901653769675"}}},
57
58 "acel:relations":{
59
60 "r1":{"acel:type":"siringWith",
```

```
61 "acel:rvmap": {"source":"1240424",
62 "cardinality":"oney2one"}},
63
64 "r2":{"acel:type":"hasMother","acel:rvmap":
65 {"source":"1576916","cardinality":"many2one"}},
66
67 "r3":{"acel:type":"hasFather","acel:rvmap":
68 {"source":"1576916","cardinality":"many2one"}}}}
```

8.2 Extraction and generation of ACEL logs: Algorithm

This section presents a detailed explanation of the Algorithm 2 presented in Chapter 5 in Section 5.3.3.

To retain only the smart contract events specified by the user during the configuration phase, we start with a filtering process. This process consists of only retaining the smart contract events specified within the configuration file as part of SC Event elements. Line 4 of Algorithm 2 shows the filtering process where the function $getMatchingEventMapping$ retrieves ev the SC Event Mapping corresponding to each eligible $scev$ smart contract event by matching its topic with the SC Event Topics contained in $SC_{events}Elements$ the list of SC Event elements within the configuration file given as input. Thus, eligible smart contract events are those for which a matching SC Event Mapping is found. Subsequently, the algorithm executes the mapping rules specified in each ev (4-21). This process unfolds as follows: the algorithm initializes e an ACEL event by assigning it an identifier (line 6) and uses the function $getEventAttributes$ to retrieve the event's attributes from the data of $scev$ in accordance with $AcelEventMapping$ the event mapping rule withing ev, and uses these attributes to populate EA the event attribute list (line 7). Additionally, it assembles O and R the lists of object and relation identifiers, respectively, associated with e by using the functions $getObjectList$ and $getRelationList$ which retrieves the value of the identifiers from $scev$ in accordance with $AcelOBjectMapping$ and $AcelRelationMapping$ the object and relation mappings contained in ev (line 8-9). For each object (line 10), the algorithm uses the function $getObjectAttributes$ to obtain $attribs$ the list of attribute mappings for the object type from $ACELObjectMapping$ (line 11). Then, it iterates over $attribs$ (line 12) and for each att checks whether the attribute is static or dynamic using function $isStatic$ (line 13). If the attribute is static it either creates a new object element or retrieves an existing one by using the function $getOrInitializeObject()$ which verifies before creating a new object if the latter was already created and exists in the log as an object element using its identifier ob. If the object does not exists it initialize

a new one and extracts its identifier from *scev* using $ev.AcelObjectMapping$ (line 14). Subsequently, it extracts for this object the static attribute's value from *scev* using function $getObjectStaticAttribute()$ and *att* (line 18). The value of the static attribute is saved in OA the object element's attribute list (line 15). After all values of the static attributes have been saved in the object element's attribute list, the object is stored in *Log* as part of the list of object elements.

Similarly, the object's dynamic attributes list is populated with dynamic attribute values from *scev* using function $getObjectDynamicAttribute$ and *att*. However, this list of values is placed in OC the list of object changes of event element e (line 18). The dynamic attributes list serves as the repository for the object's lifecycle (line 19). In line 20, the algorithm proceeds to extract the value of the object's lifecycle attribute. As this value is static and provided by the user during the configuration phase, the algorithm does not need *scev*. It only uses the function $getObjectLifecycle()$ with $ev.AcelObjectMapping$ and the object identifier *ob*. Then in line 21, it adds the lifecycle value to e's list of changes.

Analogously, the algorithm processes and stores the static and dynamic attributes of each relation within the relation static attributes list RA and the relation dynamic attributes list RC. Notably, the relation dynamic attributes list encompasses pairs denoting the target and change status of the relation. Ultimately, the event is stored in the log (line 22), and the resultant log is returned as the output (line 23).

ECOLE DOCTORALE

Titre: Découverte de processus centrés sur les artifacts à partir d'applications Blockchain

Mots clés: Processus centrés sur les artifacts, Exploration de processus, blockchain, logs d'événements centrés artifacts, GSM

Résumé:

Blockchain est une technologie de registre décentralisée et distribuée qui enregistre de manière sécurisée les transactions sur plusieurs ordinateurs, garantissant transparence et immuabilité.. La communauté BPM a reconnu son potentiel pour améliorer la gestion des processus métier (BPM) et favoriser les collaborations inter-organisationnelles. Malgré des recherches approfondies sur l'exécution des processus d'affaires basés sur la blockchain, l'exploration de données de la blockchain pour le process mining a récemment commencé à être explorée. Les études actuelles se concentrent principalement sur les processus centrés sur les activités, négligeant souvent les processus centrés sur les artefacts prévalents dans les applications blockchain. Les formats de journalisation traditionnels comme XES, bien que couramment utilisés, rencontrent des défis tels que la perte d'information et la dénormalisation lorsqu'ils sont appliqués à des données centrées sur les artefacts. L'introduction d'OCEL a partiellement abordé ces problèmes en permettant le stockage de données d'événements centrées sur les objets, mais il manque de prise en charge pour l'évolution et les relations des objets.

Cette thèse relève ces défis en proposant ACEL, une extension d'OCEL qui prend en charge de manière complète le stockage des données d'événements centrées sur les artefacts. Nous présentons une méthode centrée sur les artefacts pour recueillir des données d'événements d'applications blockchain, les convertissant en logs ACEL. La viabilité de l'approche est évaluée en utilisant les applications Ethereum Cryptokitties et Augur. Nous comparons d'abord les capacités de process mining d'ACEL avec OCEL, puis introduisons une méthode de découverte utilisant le clustering hiérarchique et l'analyse du gain d'information pour dériver des modèles GSM, la norme pour les processus centrés sur les artefacts. Notre évaluation sur Cryptokitties confirme la faisabilité de cette approche et met en évidence les avantages d'ACEL dans le process mining centré sur les artefacts.

Title: Discovering artifact-centric processes from Blockchain applications

Keywords: artifact-centric processes, process mining, blockchain, artifact-centric event logs, GSM

Abstract:
Blockchain is a decentralized, distributed ledger technology that securely records transactions across multiple computers, ensuring transparency and immutability. The BPM community recognized its potential for enhancing business process management (BPM) and fostering inter-organizational collaborations. Despite extensive research on blockchain-based business process execution, process mining from blockchain data has recently begun to be explored. Current studies mainly focus on activity-centric processes, often overlooking artifact-centric processes prevalent in blockchain applications. Traditional logging formats like XES, while commonly used, face challenges like information loss and denormalization when applied to artifact-centric data. The introduction of OCEL partially addressed these issues by enabling the storage of object-centric event data, but it lacks support for object evolution and relations.
This thesis addresses these challenges by proposing ACEL, an extension of OCEL that comprehensively supports artifact-centric event data storage. We present an artifact-centric method to gather event data from blockchain applications, converting them into ACEL logs. The approach's viability is assessed using Cryptokitties and Augur Ethereum applications. We initially compare ACEL's process mining capabilities with OCEL, and then introduce a discovery method using hierarchical clustering and information gain analysis to derive GSM models, the standard for artifact-centric processes. Our evaluation on Cryptokitties confirms the feasibility of this approach and highlights the advantages of ACEL in artifact-centric process mining.

Institut Polytechnique de Paris
91120 Palaiseau, France