A conversational AI Framework for Cognitive Process Analysis

Meriana Kobeissi

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAS025

Thèse de doctorat

A Conversational AI Framework for Cognitive Process Analysis

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

École doctorale n°626 Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 21/12/2021, par

Meriana Kobeissi

Composition du Jury :

Mme. Amel Bouzeghoub Professeur, Télécom SudParis, France	Président
M. Dirk FAHLAND Professeur, University of Technology (TU/e), Pays Bas	Rapporteur
M. Jan MENDLING Professeur, Humboldt-Universität zu Berlin, Allemagne	Rapporteur
M. Philippe MERLE Directeur de recherche, Centre Inria de l'Université de Lille, France	Examinateur
M. Bruno DEFUDE Professeur, Télécom SudParis, France	Directeur de thèse
M. Bassem HAIDAR Professeur, Université Libanaise, Liban	Directeur de thèse

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAS025

Thèse de doctorat

Université Libanaise
École Doctorale
Sciences et Technologies
Doyen

TELECOM SudParis

IP PARIS

A Conversational AI Framework for Cognitive Process Analysis

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

École doctorale n°626 Institut Polytechnique de Paris (ED IP Paris)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 21/12/2021, par

Meriana Kobeissi

Composition du Jury :

Mme. Amel Bouzeghoub Professeur, Télécom SudParis, France	Président
M. Dirk FAHLAND Professeur, University of Technology (TU/e), Pays Bas	Rapporteur
M. Jan MENDLING Professeur, Humboldt-Universität zu Berlin, Allemagne	Rapporteur
M. Philippe MERLE Directeur de recherche, Centre Inria de l'Université de Lille, France	Examinateur
M. Bruno DEFUDE Professeur, Télécom SudParis, France	Directeur de thèse
M. Bassem HAIDAR Professeur, Université Libanaise, Liban	Directeur de thèse

Acknowledgments

I would like to express my heartfelt gratitude to all those who have supported me throughout my journey to completing this thesis.

First and foremost, I thank God for His unwavering guidance and strength. The path of a thesis is not always easy, but with the supervision, support, and encouragement of numerous individuals, I have never regretted embarking on this endeavor.

I would like to thank the members of the thesis jury. I am deeply grateful to Professor Dirk FAHLAND and Professor Jan MENDLING for graciously accepting the role of thesis reviewers and for their invaluable insights and thoughtful comments. My gratitude also extends to Professor Philippe MERLE and Professor Amel BOUZEGHOUB for agreeing to be my thesis examiners.

I would like to express my appreciation and gratitude to my supervisor, Walid Gaaloul, His valuable advice, enthusiasm and constant support during this thesis allowed me to acquire new understandings and extend my experiences. Walid was not only an advisor but also a compassionate listener and a friend who consistently believed in my abilities, even when I doubted myself. I have learned invaluable lessons from Walid about being a proficient researcher and cultivating a research-oriented mindset to overcome challenges. Thank you for embarking on this journey with me; it has been an incredible experience.

I would also like to express my gratitude to my supervisor, Bruno Defude, for welcoming me into his team, and to Bassem Haidar for supervising me on behalf of the Lebanese University. Their extensive knowledge and logical thinking have been instrumental in my academic growth.

A special thanks goes to Dr. Nour ASSY, who guided me from my internship through the initial stages of my thesis. Her unwavering support and mentorship provided me with a strong foundation as a researcher, allowing me to chart a path toward achieving my goals.

I owe a debt of gratitude and warm affection to the members of the Computer Science Department at Telecom SudParis. Special thanks to my office partner and dear friend Imen JERBI for the delightful moments we shared.

My deepest appreciation goes to my family: my mother Abir, my sister Diana, my brother Mohamad, and my brother-in-law Moslem, who always provided words of encouragement whenever I faced self-doubt. I also thank my late father, who remains with me in spirit. To my loving fiance, Rami, I am eternally grateful for your love, encouragement, understanding, and support during challenging times. I cannot forget our youngest family member, my beloved niece Hayam. You supported me without even knowing it. I love you so much. I dedicate this thesis to all of you, my wonderful family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Context & Motivation

In the constantly evolving environment of contemporary organizations, Business Process (BP) serves as the foundational structure for reaching specific targets and objectives. These meticulously organized tasks encompass a variety of assignments, collaborations, and workflows, which are essential for the seamless operation of various sectors and industries. From taking care of customer orders to providing services, from initiating product development to overseeing financial management, these BPs are pivotal to successful operations.

As technology continues to evolve, these business procedures have been transforming into increasingly data-centric operations. Every assignment and collaboration generate a stream of data, often referred to as event logs. These logs create a digital record of BP execution, capturing a variety of details such as the order of activities, timestamps, resources used, and other relevant data. This data is typically tracked by various information systems, such as customer relationship management systems, enterprise resource planning software, and other digital platforms that gather data from numerous process execution touchpoints.

Table 1.1 shows an example of an event log pertaining to a loan application process. The table enumerates the sequence of activities that unfold during the life cycle of each loan

Application ID	Activity	Timestamp	Resource	Offer ID
1	Application Submitted	2023-01-01 09:00:00	WebApp	N/A
1	Application Accepted	2023-01-01 11:00:00	Agent A	N/A
1	Offer Created	2023-01-01 12:00:00	Agent A	O1
1	Offer Created	2023-01-01 12:10:00	Agent A	O4
1	Offer Sent	2023-01-01 12:30:00	Agent A	O1
1	Offer Sent	2023-01-01 12:40:00	Agent A	O4
1	Offer Accepted	2023-01-02 15:00:00	Customer	O4
1	Loan Disbursed	2023-01-03 10:00:00	Agent A	O4
2	Application Submitted	2023-01-02 08:00:00	Mobile	N/A
2	Application Accepted	2023-01-02 10:00:00	Agent B	N/A
2	Offer Created	2023-01-02 11:00:00	Agent B	O2
2	Offer Sent	2023-01-02 11:30:00	Agent B	O2
2	Offer Accepted	2023-01-02 15:00:00	Customer	O2
2	Loan Disbursed	2023-01-03 11:00:00	Agent B	O2

Table 1.1: Example Event Log Table

application, focusing here on two exemplar cases: Application 1 and Application 2. Each row in the table represents an event, characterized by several attributes. The 'Application ID' column uniquely identifies each loan application, aiding in the isolation of the sequence of activities related to a specific application. The 'Activity' column specifies the various stages each loan application goes through, such as submission, application acceptance, offer creation, and disbursement of the loan. Timestamps are recorded in the 'Timestamp' column, which provides temporal information on when each activity was executed. This data is crucial for understanding the order of activities and for potential time-based analysis such as bottleneck identification or compliance checking. The 'Resource' column indicates the entity—be it an automated system or a human agent—responsible for executing each activity. Lastly, the 'Offer ID' column associates specific loan offers with activities where applicable. For instance, in Application 1, two offers (O1 and O4) were created and sent, but only one (O4) was accepted and disbursed. In contrast, Application 2 involves just a single offer (O2). It's worth noting that this event log could be extended to include additional attributes to enrich the analysis further. For instance, associated data such as the requested loan amount for each application, the offered amount, monthly costs associated with each offer, and other financial or risk metrics could be included.

Process analysis, a discipline aimed at harnessing the potential of these event logs, focuses on extracting valuable insights [121]. By meticulously studying and decoding process-related data, this practice offers organizations a comprehensive understanding of their operational procedures. It also identifies deviations from intended models and suggests areas for improvement. This data-driven approach empowers organizations to make informed decisions and enhance their operations continually. This analytical process can be approached from either the instance (looking into instances of process executions) or the process level (examining the overall process). Each of these operationalizes different analytical techniques, engages with various data attributes, and generates unique insights that contribute to an organization's

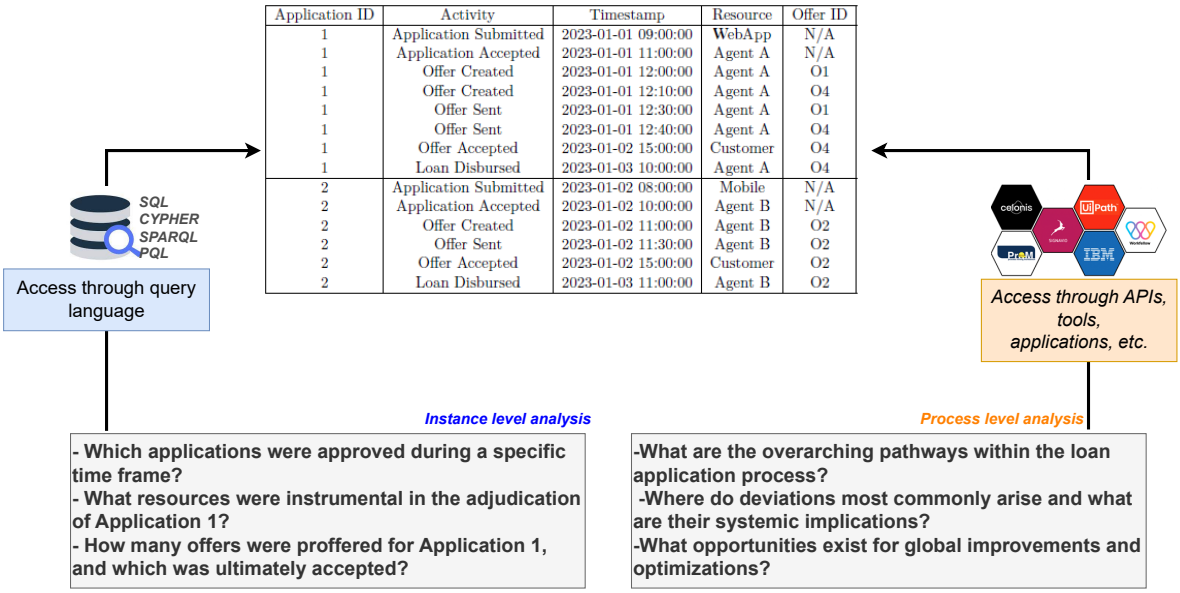


Figure 1.1: Example of instance level and process level analysis questions

strategic decision-making framework. Instance-level analysis is fundamentally concerned with the granular examination of individual events within a process. Such an analysis involves a meticulous investigation of the characteristics of each event, such as the nature and sequence of activities, associated timestamps, involved resources, and other related attributes. In contrast, process-level analysis provides a broader perspective by employing APIs and methods on the process data.

Figure. 1.1 illustrates examples of analysis questions tailored to both instance and process levels. For instance-level analysis, the storage and representation of process data must align with specific technology, necessitating queries crafted in a particular query language. Conversely, process-level analysis requires users to analyze process execution data through the application of specialized methods, tools, and APIs. Consequently, to analyze processes effectively, users must either construct corresponding query languages or access the appropriate APIs and methods. Both scenarios demand users to possess a minimum level of technical expertise presenting a substantial challenge and limitation, hindering users from effectively analyzing process execution data and reaping the benefits of process-related insights.

Initially, the essence of instance-level process data analysis centers on the task of querying information from the stored process data. Process querying, a foundational component within the broader realm of process analysis, provides users with the capability to interactively retrieve and manipulate process-related data using a variety of tools and query languages [121]. This field encompasses methods aimed at the automated manipulation of repositories containing models describing observed or projected processes. A process querying method, within this context, denotes a systematic technique that, given a repository of processes and a specific process query, executes the query on the provided repository. This repository, in turn, comprises an array of models encompassing elements like behavioral models, simulation mod-

els, correlation models, and even process execution data such as event logs. A process query, on the other hand, functions as a formal directive to oversee the management of this repository, often pertaining to Create, Read, Update, and Delete (CRUD) operations. The realm of existing works within process querying can be classified based on two key dimensions: the nature of input data, which pertains to the models stored within the process repository, and the ultimate querying objective, encompassing the specific CRUD operations applied to manipulate the repository's contents. For instance, studies such as those conducted in [17, 10] focus on querying behavioral models to glean insights into structural topology and process attributes. These studies utilize dedicated querying languages like BP-QL [17] and BPMN-Q [10] to facilitate the extraction of valuable information from these models.

An important subfield within process querying is querying process execution data. Within this context, the process repository primarily stores process-related execution data that can be accessed and manipulated via specialized query languages. Various approaches convert this process data into specific formats or storage representations, facilitating querying through languages such as FPSPARQL, SQL, NoSQL, and Cypher, etc. [20, 168, 58]. This enables users to tailor queries to their specific requirements, thereby extracting pertinent information for in-depth analysis. For instance, consider a process repository storing event data related to the patient care process within a hospital. Within this context, a process query could entail adding a new patient care process to the repository (a Create operation) or identifying sequences of activities related to the treatment of specific patients (a Read operation).

While the potential benefits of process querying are considerable, it is worth noting that the complexity of certain query languages like SQL, Cypher, and SPARQL can serve as a barrier to entry for some stakeholders. Business users or professionals, who may not possess the technical expertise to navigate these languages, might constrained in their ability to fully leverage the power of process data. For instance, a hospital administrator, who is not familiar with FPSPARQL language, might struggle to formulate a query that identifies potential bottlenecks in patient flow through different hospital departments, thereby missing opportunities for operational improvement.

Next, when engaging in process-level analysis of process data, the approach entails the utilization of methods and algorithms that harness process execution data extracted from information systems. These methodologies, collectively referred to as process mining techniques [156], serve the purpose of revealing, monitoring and improving real-world processes. With the help of process mining techniques, organizations can glean valuable insights into process behavior, spot inefficiencies, detect compliance violations, and identify opportunities for process optimization. The families of process mining techniques include discovery (finding the model of the processes), conformance (comparing the expected process model with the actual process execution data), enhancement (extending or improving existing process models), and prediction (forecasting future process behaviors).

Despite the importance of process mining methods, data scientists and process analysts often encounter significant challenges when attempting to apply these methods in practice:

- **Challenge related to the discoverability of process mining methods.** One of

the primary difficulties is related to the discovery and selection of an appropriate method that can meet their specific needs and requirements. For instance, consider an analyst who is interested in monitoring and predicting the completion time of processes using a machine-learning approach. The analyst is confronted with a considerable number of prediction methods available in the literature, each with unique properties, such as the type of predictions they make, the employed algorithms, the required inputs and configuration parameters [47]. As a result, searching for a suitable method, that satisfies the analyst's needs, can be time-consuming involving an exhaustive and manual evaluation of unstructured descriptions.

- **Challenge related to the applicability and integration of process mining methods.** The second challenge associated is related to their accessibility and applicability. Typically, process mining methods are available either as source code or implemented in separate software tools. In recent years, the field of process mining has been supported by several open-source projects, including ProM [157], bupaR [72], PM4Py [27], RapidProM [103], Apromore [90], as well as commercial tools such as Disco [67], Celonis, and Everflow, etc. These tools offer a wide range of functionalities, such as log data preprocessing, process discovery, conformance checking, etc. However, accessing a method through its source code can be daunting for analysts lacking proficiency in the programming language used for implementation or technical expertise to understand the method's technical details. Additionally, certain software tools that offer process mining methods might be not compatible with other software applications, severely limiting their applicability across diverse workflows and their integration into custom applications.

A leading strategy to address the challenges associated with process analysis tasks at the instance level (i.e. querying process execution data), and at the process level (i.e. the application of process mining methods and APIs) is the incorporation of cognitive capabilities [15]. These capabilities leverage various artificial intelligence mechanisms that are designed to simulate human cognitive faculties. These solutions include but are not limited to natural language processing (NLP), text analytics, machine learning, large language models (LLMs), etc. The implementation of such capabilities serves to augment the interface between human users and computational systems. Incorporating such capabilities into process analysis has the potential to bridge the gap between business users, data analysts, and process analysts, allowing them to harness the power of process data for decision-making and process improvement.

The context of this thesis revolves around AI-driven solutions geared towards facilitating analyzing processes using their execution data and specialized methods and techniques. The primary objective is to provide solutions that make the analysis tasks more intuitive and user-friendly enabling users to effortlessly analyze their BPs. This is achieved through the integration of cognitive capabilities, as illustrated in Figure 1.2.

This, in turn, significantly improves the user experience with process analysis by:

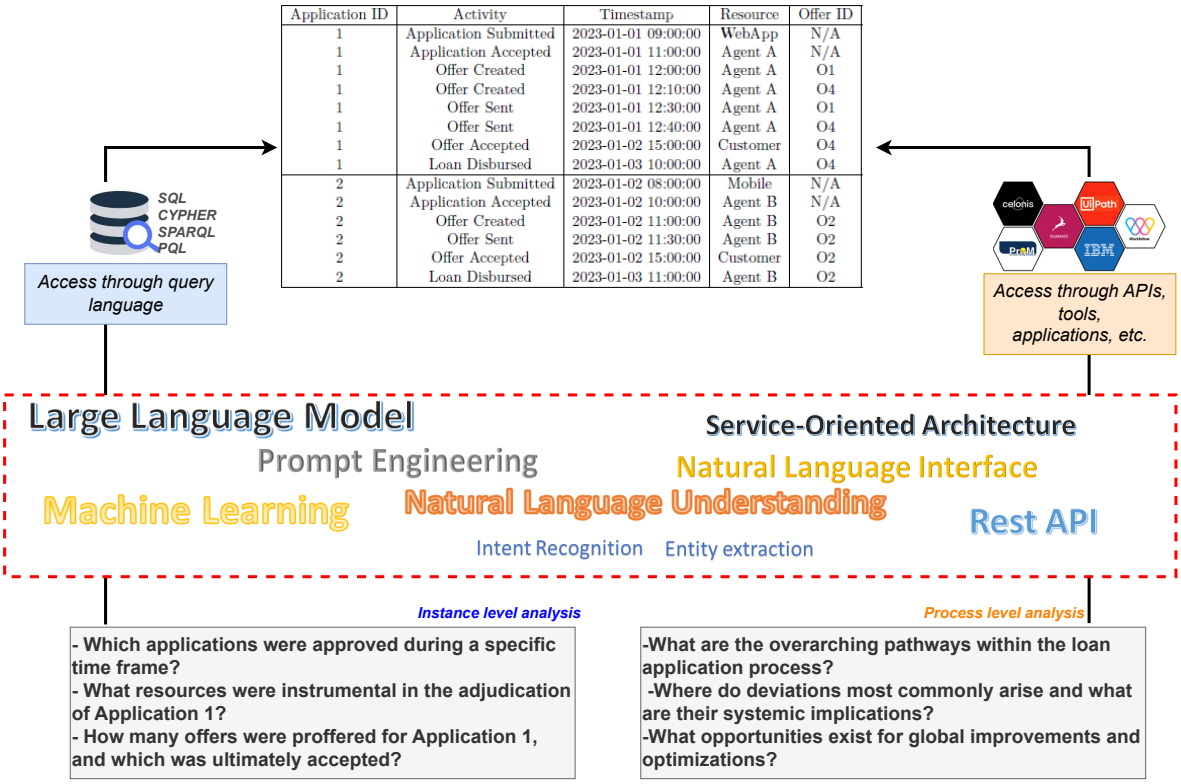


Figure 1.2: Incorporating AI solutions to facilitate process analysis

- **Enabling universal accessibility:** Making process-related data accessible to users across all levels of expertise. This democratization of data empowers not just data analysts, but also business users and process analysts to glean insights from the process data.

- **Enhancing analysis simplification:** The process of analysis is streamlined through the integration of Natural Language Interfaces (NLI) and cognitive support. This eliminates the necessity for intricate query languages, specialized technical knowledge, or familiarity with specific tools. As a result, users with diverse technical backgrounds gain the capability to effectively analyze their processes without hindrance.

- **Efficiency and automation:** Automation of routine analysis tasks through cognitive tools reduces manual workload, freeing up experts to focus on strategic initiatives and value-added activities.

While AI solutions offer a promising avenue for innovation, their integration into process analytics also presents a set of challenges that form the basis of this thesis. In what follows, we discuss some of these research problems in Section 1.2. We outline then our thesis objectives, principles, and contributions in Section 1.3. Finally, we present the structure of the thesis in Section 1.4.

1.2 Research Problem: How to incorporate cognitive capabilities to facilitate process analysis tasks?

As highlighted in Section 1.1, the realm of process analysis is diverse and can be approached from either the instance or the process level. However, applying process analysis in practice demands users to possess a certain level of expertise and technical knowledge. This requirement limits the applicability of process analysis and hinders users from fully benefiting from valuable insights within process data. Hence, there is a pressing need for solutions that enhance the accessibility of process analysis for human users by integrating AI techniques and cognitive capabilities to create cognitive process analysis. In light of this, we have identified two key research challenges critical to realizing cognitive process analysis. The initial research challenge revolves around making the querying of process-related data an intuitive task that caters to users with diverse levels of expertise. This entails using Natural Language (NL) to eliminate the need for complex query languages or specialized technical knowledge. Consequently, the first research question is **(RQ1) How to make process data accessible to human users in a natural manner?**. The second challenge focuses on simplifying the accessibility and usability of process mining methods for high-level analysis. This involves making these methods easily discoverable and accessible to analysts, aligning with their specific needs described in a natural way. Consequently, the second research question is **(RQ2) How to make the discovery and accessibility of process mining techniques intuitive and in a natural manner?**

Within the scope of **RQ1**, our research challenge is to infuse cognitive capabilities into the interaction between users and process data. Our primary focus lies in automatically generating structured queries from NL, making data querying tasks more intuitive and inclusive for a wider range of users. Meanwhile, **RQ2** directs our efforts towards incorporating cognitive capabilities to streamline the process of discovering and accessing process mining techniques specific to user's needs.

These research questions can be further divided into several sub-questions. Some of them were previously discussed in related works and others were not previously handled. We separate these research sub-questions according to RQ1 (Section 1.2.1) and RQ2 (Section 1.2.2).

1.2.1 How to make process data accessible to human users in a natural manner?

The core research challenge is to ensure that process execution data is readily accessible and queryable for all users, removing the need for specialized query languages or technical expertise. Existing process query languages are typically designed for data scientists, presupposing a certain level of technical familiarity with process schemas, and query languages such as SQL, Cypher, and SPARQL. However, a significant oversight in current approaches is their failure to make process data accessible to domain analysts (e.g., those in healthcare or insurance). More crucially, a prominent limitation of existing process querying technologies is their inability to offer natural access to process data for human users. Consequently, the first challenge revolves around finding solutions that can automatically generate the corresponding structured query from NL, allowing users to query process data in a way that feels intuitive and user-friendly.

Additionally, the proposed solution must possess the necessary generality to seamlessly transition between diverse process domains with minimal manual intervention. Process data, intricately tied to specific domains, assumes unique characteristics that distinguish one process domain from another. For instance, consider the distinct nature of process data in the healthcare domain, where patient treatments, diagnoses, and medical interventions shape the events recorded. In contrast, process data in the context of supply chain management could revolve around order placements, inventory updates, and shipping activities. This domain specificity underlines the challenge of devising a querying solution that minimizes the manual effort required for domain adaptation. A robust querying solution must incorporate automated mechanisms that infer domain-specific characteristics and adjust query processing accordingly. This is essential to ensure the applicability of any proposed solution to various domains.

Moreover, the automated solution must be aligned with a particular storage technique employed for storing process data and be designed around a specific query language to streamline the query construction process. This necessitates the careful selection of an efficient storage technique and an appropriate query language tailored to the task of querying process data. Process execution data captured by information systems can come in either single or multi-

dimensional formats. In a single-dimensional context, events are linked with a solitary entity identifier (case ID), which allows for correlation centered around a specific entity. This facilitates the examination of events from a particular vantage point. Conversely, in most cases, processes exhibit multi-dimensional characteristics. In such scenarios, the conventional concept of a case ID might not be applicable. Instead, data are grouped into objects, and each event can be tied to one or more of these objects. This setup enables the simultaneous analysis of events from multiple viewpoints. Moreover, these objects can establish relationships with each other, opening the door for correlations based on combinations of objects. Therefore, for effective access and manipulation of process execution data, suitable storage and querying mechanisms are imperative. The research outlined in [55] offers insights into the concepts and requirements for modeling and querying multi-dimensional process data. We distilled these requirements into two principal facets for querying process data:

- Since event data has inherent connections, the storage approach should efficiently model relationships between (i) events, (ii) events and objects, and (iii) objects as core components.
- Querying paths should be straightforward and efficient.

Various strategies have been proposed to facilitate the querying of process data stored in files (using standardized formats like XES) [124, 168], relational databases [139, 113, 48], or graphical models [20, 55] (for an overview, refer to [55]). Within this context, the initial hurdle lies in proposing an effective model for representing and querying process data.

Given the limitations highlighted earlier, we have formulated the following sub-questions for exploration:

- **RQ1-1**: How to provide an AI-based solution that automates the process of querying process data using NL?
- **RQ1-2**: What design principles can be implemented to create an approach that possesses sufficient generality, allowing for seamless transitions between diverse process domains with minimal manual intervention?
- **RQ1-3**: What storage technique and query language should the automated solution be designed around to ensure effective and efficient querying?

1.2.2 How to make the discovery and accessibility of process mining techniques intuitive and in a natural manner?

In light of the intricate and varied landscape of process mining methods, challenges arise when attempting to offer all-encompassing solutions for their discovery and accessibility. With the proliferation of methods spanning a diverse spectrum, the seamless discovery and access to the most pertinent techniques tailored to individual needs becomes an intricate task. In

the following, we delve into the primary limitations associated with developing solutions to enhance the discovery and accessibility of process mining methods.

The first limitation pertains to the varied ways in which process mining methods are implemented. These methods manifest in various forms, from source code written in different programming languages to integrated features within specialized process mining tools or as standalone applications. This diversity in implementation poses several challenges. Firstly, it can limit the accessibility of these methods based on a user's expertise with a particular programming language or tool. Secondly, it can hinder the seamless integration of these methods into broader software applications, leading to potential compatibility and interoperability issues. For instance, a method implemented in Python might not easily integrate with a system built on Java. This lack of standardization in implementation can stifle innovation and limit the broader adoption of process mining techniques. To address this, there is a need for solutions that streamline the integration and accessibility of these methods. Such solutions should ensure that users can effortlessly access and integrate these methods, irrespective of the programming language in which they are implemented or other potential barriers.

The second limitation stems from the absence of standardized documentation that provides a unified description of all methods across the diverse fields of process mining. Given the expansive array of techniques available in areas such as discovery, conformance checking, prediction, and enhancement, it becomes evident that each field has its unique set of methods, each characterized by specific properties. For instance, while process discovery methods might be described based on the algorithm used for model discovery or the notation employed for modeling the resultant model, prediction methods might be characterized by the type of prediction, the methodology employed, and the nature of the predictor model. This lack of a standardized approach to documentation means that knowledge about these techniques remains fragmented and siloed. While several recent surveys, such as [9], [147], and [140], have attempted to shed light on specific sub-domains within process mining, they often focus on niche areas. Even when multiple surveys target the same domain, they might characterize methods using distinct properties, leading to inconsistencies. This fragmented landscape underscores the pressing need for a standardized and unified representational model or description that encompasses all methods across different fields of process mining. Such a unified approach would provide comprehensive information about the properties and nuances of these methods, aiding users in understanding and selecting the most appropriate techniques for their needs.

Furthermore, it is imperative that any proposed solution addresses the aspect of user-friendliness. This entails ensuring that the solution is designed to be intuitive and straightforward, accommodating users of all backgrounds in effectively discovering and applying process mining methods.

In light of these limitations, our second research challenge revolves around creating a more cohesive and accessible ecosystem where process mining techniques are readily discoverable and accessible, irrespective of the underlying technical intricacies or the user's level of expertise. We intend to establish a unified modeling and representation framework for the spectrum of available process mining methods. This framework would encapsulate essential

method properties, providing users with a comprehensive and standardized view of each technique's capabilities and applicability. In addition, we aim to provide users with a user-friendly solution allowing them to easily discover and access these techniques. To address this research problem, we need to answer the following questions:

- RQ2-1: How can the accessibility and integration of process mining methods be streamlined?
- RQ2-2: Which properties should be used to characterize available methods to facilitate seamless discovery?
- RQ2-3: How to provide a user-friendly solution that assists users in discovering and accessing methods that cater to their individual requirements?

1.3 Thesis principles, objectives and contributions

1.3.1 Thesis principles

In this thesis, we consider the following principles:

> **Principle 1: User-Centric Design:** At the heart of our approach lies the end-user (business process users, analysts, data scientists, etc.). Every solution, tool, or methodology we develop will prioritize user experience, ensuring that even those without technical expertise can effortlessly interact with the provided solution;
>
> **Principle 2: Automation:** The proposed approaches should automate intricate tasks that users would otherwise need to perform manually;
>
> **Principle 3: Adaptability & Generality:** Recognizing the dynamic nature of process domains, our methodologies will be designed to adapt to diverse domain-specific characteristics. This ensures that our solutions remain relevant and effective across a myriad of application scenarios;

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 real datasets such as real process execution data with user's collected data and feedback. Therefore, the implementation, experiments, and case study results with end users should be detailed.

1.3.2 Thesis objectives

The overarching ambition of this thesis is to harness the potential of incorporating AI solutions to facilitate process analysis tasks. Grounded in the challenges and research problems

delineated in the preceding sections, we consider two main objectives in this thesis. The first objective revolves around the automated querying of process data from NL. This encompasses the creation of user-centric interfaces and tools that not only render process data accessible but also make it intuitive for a wide range of users. The second objective is centered on making the process mining methods easily accessible and discoverable. Our aim is to establish a unified and user-friendly environment where process mining techniques are readily discoverable and accessible, irrespective of the technical intricacies involved or the users' level of proficiency.

Within the scope of the first objective, we delineate three sub-goals. Firstly, there is a fundamental need to offer an automated solution that streamlines the querying of process data through NL. Secondly, we aim to create a solution that exhibits adaptability, smoothly transitioning between various process domains with minimal manual effort. Lastly, the automated solution should be constructed in alignment with efficient storage techniques and query languages.

In pursuit of the second objective, we chart out an additional set of three sub-goals. Firstly, our focus shifts towards devising solutions that promote the smooth integration and accessibility of process mining methods. Secondly, we aim to establish a standardized method description framework that facilitates the discovery of methods based on their characteristics. Lastly, we endeavor to elevate the user experience by crafting user-friendly interfaces that enable the effortless discovery and utilization of process mining methods based on natural requirements. Through these sub-goals, we aspire to advance the accessibility, and usability of process mining methods for a diverse user.

Our objectives are summarized as follows:

- **Objective 1:** Facilitate process data querying by automatically constructing structured queries from NL;

  This objective requires the achievement of three sub-objectives:

  - **Objective 1.1:** Put forth an AI-driven solution for automating the construction of queries over process execution data using NL;
  - **Objective 1.2:** Develop the querying solution adaptable to various domains, reducing the effort needed when transitioning between different process domains;
  - **Objective 1.3:** Choose an efficient storage technique and query language as the foundation for designing the automated solution;

- **Objective 2:** Facilitate the discoverability and accessibility of process mining methods based on the requirements expressed in NL;

  This objective requires the achievement of three sub-objectives:

  - **Objective 2.1:** Provide a service-oriented solution that streamlines the integration of process mining techniques;

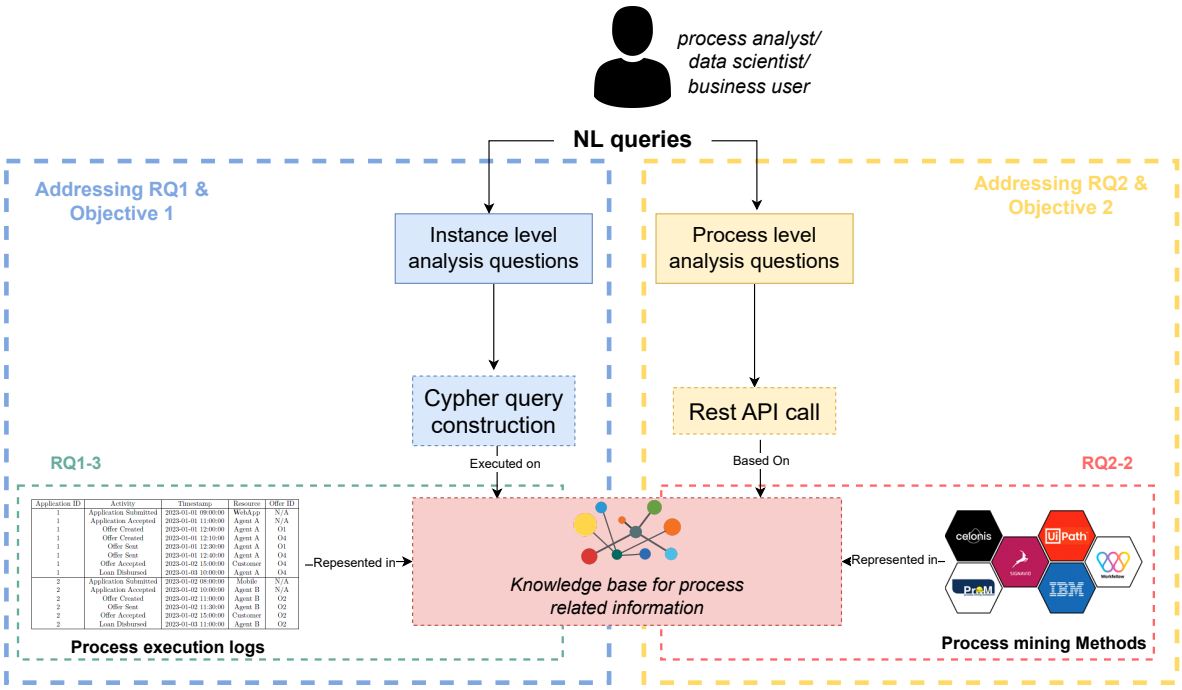


Figure 1.3: Two complementary approaches toward a conversational AI for cognitive process analysis

- **Objective 2.2:** Establish a standardized description metamodel for process mining methods, including their properties, inputs, and outputs, to enable effortless discovery;
- **Objective 2.3:** Enhance the user experience by automating the discovery and invocation of methods that align with user requirements expressed in NL;

1.3.3 Thesis contributions

To meet the above objectives while handling the described research issues, we introduce two complementary approaches toward a conversational AI for cognitive process analysis. These approaches revolve around querying and accessing a knowledge base containing process-related information, as illustrated in Figure 1.3. The first approach focuses on simplifying the cognitive querying of process data by automatically generating a database query from NL queries. This approach is highlighted by the blue pointed area in Figure 1.3 and directly tackles research problem **RQ1** and **Objective 1**. This approach encompasses three primary contributions, which are further described below:

- **Graph Metamodel for representing process execution data:** We propose a graph metamodel rooted in the principles of Labeled Property Graph (LPG) [7] for the effective storage of process data (highlighted by the green pointed area in Figure1.3). The choice of LPG is motivated by its capacity to explicitly represent connections within process data. This metamodel encompasses various node types, relation types, and properties,

facilitating the storage of diverse concepts and relations linked to multi-dimensional process data. Additionally, we employ the Cypher language[1] to access and retrieve data. The Cypher query language's aptitude for efficiently querying subgraphs and paths aids in addressing **RQ1-3** and aligning with **Objective 1.3**.

- **Natural language interface for querying process data:** We introduce a pipeline based on NLP to guide end users in querying process data. Specifically, we focus on the read operations to retrieve data and answer process-related questions. The user-facing interface of this pipeline receives user queries formulated in NL and automatically constructs the corresponding Cypher queries for execution over structured process data. This hybrid pipeline combines the strengths of both machine learning and rule-based methodologies. The process unfolds in two key stages: First, we employ machine learning techniques for intent detection and entity extraction, leveraging a model trained on a collection of NL queries paired with their respective intents and entities. To align with **Principle 3** and **Objective 1.2**, we propose general patterns for intents and entities, that are defined based on the LPG meta-model in general and that are instantiated according to the event property graph model. Subsequently, in the second stage, a rule-based mechanism generates the Cypher query based on the intent and entities identified in the previous stage, addressing **RQ1-1** and aligning with **Objective 1.1**.

- **Automated generation of NL training data:** As previously mentioned, the NLI compromises a machine learning model that should be trained with a set of NL training data. However, these NL queries are intricately tied to specific process domains, necessitating the generation of new training data when transitioning between domains. To align with **Principle 3**—minimizing the effort required for multi-domain adaptation—we propose an automated approach leveraging prompt engineering and LLMs for the generation of NL queries. By harnessing the capabilities of LLM like GPT, we can automatically produce a diverse range of NL queries without relying on manually annotated datasets for each domain transition, thus addressing **RQ1-2** and **Objective 1.2**.

The second approach focuses on improving the discovery and accessibility of process mining techniques, effectively addressing **RQ2** and aligning with **Objective 2**. This approach is highlighted by the yellow pointed area in Figure 1.3. It involves the description of process mining techniques and introduces a service-oriented solution specifically crafted to invoke the appropriate API of process mining methods that fulfill the analysts' requirements. This approach embraces a service-oriented paradigm, presenting process mining techniques as services. It successfully tackles **RQ2-1** and aligns with **Objective 2.1**. Central to this framework is the utilization of Rest APIs for process mining services, a choice that streamlines the integration process. The architecture comprises three key components.

The first component, "Service Description", utilizes a graph metamodel based on LPG to describe the available discovery, conformance, and prediction methods (highlighted by the red pointed area in Figure1.3). This metamodel encompasses essential concepts, such as service

[1] https://neo4j.com/developer/cypher/

properties, required inputs and outputs, and submodules or micro-functionalities constituting the services. This comprehensive representation directly addresses **RQ2-2** and aligns with **Objective 2.2**.

The second component, "Unified Service-Oriented REST API Design", capitalizes on the inherently functional nature of process mining methods. It presents users with a cohesive and comprehensive Rest API design tailored to discovery, conformance, and prediction services. This design encompasses meticulous considerations, including URL structure, HTTP method selection, and response code specifications.

The final component, "Services Matching", addresses **RQ2-3** and aligns with **Objective 2.3** by automating the process of matching users' requirements with suitable process mining services. This eliminates the need for time-consuming manual searches in unstructured data. This component operates through two key phases. It commences by querying the process mining methods graph to identify methods aligned with the user's specifications. This is achieved by automatically crafting Cypher queries based on provided NL queries, a process enhanced by the capabilities of LLMs like GPT-4. Once suitable methods are identified, users can make selections from the available options. Upon selection, the second stage generates corresponding REST API calls utilizing the REST API design outlined in the second component. This two-phase mechanism provides a streamlined and user-centric process for accessing and employing process mining methods **(Principle 1)**.

The two solutions were implemented as standalone applications. The initial solution incorporates an NLI that interacts with users to gather NL queries. To validate its effectiveness, the solution was rigorously evaluated using two publicly available process datasets. Additionally, over 520 NL queries were collected from external users, with the aid of a paraphrasing tool. Furthermore, we embarked on a proof-of-concept evaluation of our service-oriented architecture. We used more than 110 NL queries to evaluate the service matching component. These NL queries are specifically tailored to descriptions of process mining services. Additionally, we carried out a use case study with external participants to gauge user experience in searching and accessing process mining methods and to gather feedback. Our experimental outcomes have been made publicly accessible, ensuring transparency and reproducibility within the research domain. This availability encourages further investigation and analysis, facilitating practical insights for future research endeavors.

1.4 Thesis outline

This thesis is structured as follows. Chapter 2 lays out the background and pertinent works relevant to our study's context. We initiate by presenting an overview of works concerning process querying and process data storage, offering insights into the current research landscape. Subsequently, we delve into studies focused on integrating NLP into database systems. Additionally, we examine initiatives that harness AI and cognitive capabilities to simplify intricate analytical tasks. Our exploration extends to scholarly endeavors dedicated to enhancing the accessibility and discoverability of process mining methodologies. Lastly,

we provide a comprehensive discussion of research related to the description and alignment of web services.

Chapters 3, 4, 5 are the core of our thesis which elaborate our main contributions. Chapter 3 presents the graph metamodel we devised for representing multi-dimensional process data. Chapter 4 showcases our NLI for querying process execution data. This section also highlights our automated approach for generating NL queries, which supports training the first component of our interface. Chapter 5 delves into the service-oriented architecture for discovering and accessing process mining techniques with its three main components: service description, unified service-oriented Rest API design, and service matching.

Chapter 6 shed light on potential avenues for furthering our research, pinpointing areas ripe for enhancement and broadening within the scope of our contributions. It offers a thorough analysis of prospective improvements and augmentations for each facet of our work.

Finally, Chapter 7 concludes this thesis by summarizing the work presented.

Chapter 2

Background & Related Works

Contents

2.1 Introduction

In this chapter, we embark on an in-depth review of relevant literature and studies that anchor our research thesis. Initially, Section 2.2 delves into research pertaining to process querying techniques and various methodologies for storing process data which is directly related to research question **RQ1-3**. Next, the landscape of NLP as applied in databases and technical Domains is mapped out in Section 2.3. More specifically, Section 2.3.1 details the current strides in crafting NLIs tailored for querying database systems. Further advancing

in this domain, Section 2.3.2 elucidates the methodologies related to tasks of intent detection and entities extraction—central elements in the realm of conversational agents. In the next section, we highlight studies closely related to incorporating cognitive capabilities into process analysis tasks. Within this domain, Section 2.4.1 sketches the current state of process analysis concerning unstructured data. This is complemented by Section 2.4.2 which elucidates the techniques enabling NL queries over process data and their integration with process mining tools. The potential of leveraging LLMs for intricate process analysis tasks is showcased in Section 2.4.3. The last two sections transition our focus to research related to our third contribution, which revolves around a service-oriented architecture designed for the discovery and access of process mining techniques. In Section 2.5 we offer an overview of existing research endeavors aimed at enhancing the discoverability (Section 2.5.1) and accesibility (Section 2.5.2) of process mining techniques. Finally, in Section 2.6 we discuss existing approaches concerning services description (Section 2.6.1), and approaches related to automated web service discovery (Section 2.6.2).

2.2 Process Querying and Data Storage

A process querying method is a technique for managing a process repository by applying CRUD operations [122]. A process repository stores a model that describes a specific process (e.g. process model, event log). Existing work on process querying can be classified according to the input data (i.e. model stored in the process repository) and the querying goal (i.e., applied CRUD operations). The first category is concerned with querying process models. These works suggest a query language for retrieving information or updating an existing process model (e.g. [78, 41, 17]). They also concentrate on efficient querying of process models (e.g. [120, 76, 11]). The second category addresses the problem of querying execution traces of business processes at run-time (used for monitoring) or post execution (e.g.[16, 43, 44, 42, 112]). Researches in this category use as input a process model and its event log. Finally, the last category focuses on querying execution traces of business processes in the form of event logs only (e.g. [20, 124, 168, 58]). In our work, we focus on the third category for querying process execution data.

Over the years, various models and storage techniques have been developed to store process execution data. One of the primary methods is file storage, where process data is sequentially stored in files. These files can be in diverse formats, including CSV, XES, and OCEL. Specifically, XES [71] and OCEL [61] have emerged as the two primary standards for storing event logs in the realm of process mining. Process execution data is multi-dimensional and object-centric in nature [1], this refutes the appropriateness of sequential storage of event data in the form of XES event logs (e.g. [124, 168]). The XES standard groups events under a single case notion and it does not allow to modeling multi-dimensional process data, where each event is related to one or more case notions. On the other hand, OCEL has been introduced as a new standard to represent this multi-dimensional event data. While OCEL enhances event-object correlation, querying complex relationships between entities in an OCEL representation can be challenging. For instance, querying path between related events across multiple objects

may not be straightforward.

Relational data model has been used by the majority of querying techniques to allow the representation of one-to-many and many-to-many relations between events and cases. For instance, the research [139] introduces a mining approach that operates directly on relational event data by querying the log using conventional SQL. By leveraging database performance technology, the approach allows the detection of specific control-flow constraints. Another research, [132], showcases a framework that employs SQL queries on relational process data. This framework addresses multiple declarative process mining scenarios, including process discovery, conformance checking, and query verification. The paper [48] tackles the intricacies of conducting process mining on extensive datasets. It introduces a specific database operator that identifies the 'directly follows' relationship, a core concept in process mining. Furthermore, the study explores the operator's equivalence attributes, which are instrumental in enhancing query efficiency. The work [113] introduces a meta model that seamlessly integrates both process and data perspectives. The goal is to generate diverse views from a database in a highly flexible manner. This integration is pivotal for deriving meaningful insights from process mining endeavors. Although relational data modeling is a prevalent approach, the absence of explicit storage of relationships between events complicates the querying of paths. These complications or inefficiencies can adversely impact the overall querying process.

Graph-based data models, namely RDF [130] and labeled property graphs [7], have been proposed to overcome the limitations of relational modeling and querying. The inherent design of graph data models emphasizes relationships, enabling more intuitive representation and querying of paths between interconnected events. This design sidesteps the complexities and overheads tied to executing JOIN operations across multiple SQL tables. The pioneering graph-based model for event data storage utilized RDF [20, 18, 19], where events are depicted as nodes, and the relationships between events and entities, as well as inter-entity relations, are illustrated as edges. Such graphs can be efficiently queried using SPARQL. In the study [20], the authors put forth an enhanced version of SPARQL tailored for querying event data housed in RDF graphs. Another research, [18], introduces a specialized model tailored for encapsulating process knowledge. This model captures process-centric entities, abstractions, and their interrelations in a graph format. The model is equipped with tools for discovery, extraction, and analysis of process data, adeptly translating process-focused queries into graph-centric ones. Additionally, the paper [19] delineates a model for process OLAP (P-OLAP), spotlighting OLAP-centric concepts within the process landscape, such as process cubes, dimensions, and cells. This research also unveils a graph processing engine, grounded in the MapReduce paradigm, optimized for large-scale analytics on process graphs.

LPG serves as another graph-based approach to store event data. Compared to RDFs, LPGs offer a more compact graph size since they permit the storage of key/value pairs directly within a node or relationship. In the context of financial auditing, the study [161] represents behavior across two entity types as a graph, detailing the directly-follows relationship for each entity or relation. Similarly, the research [24] transforms object-centric logs into two distinct graphs: one illustrating the correlation between events and entities, and the other showcasing

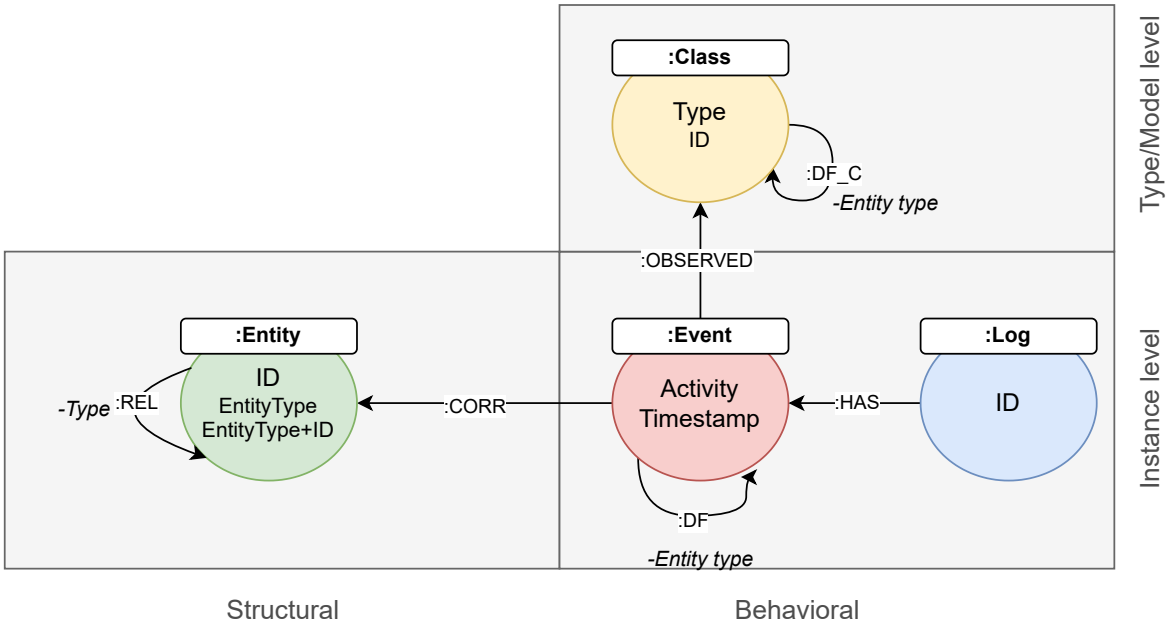


Figure 2.1: Graph schema proposed by [55] for storing multi-dimensional process data

the directly-follows relationship between any pair of events per entity.

In this thesis, we propose a graph metamodel based on LPG to store concepts and relations related to multi-dimensional process data (detailed in Chapter 3). Our metamodel draws inspiration from the work presented in [55] which proposes a comprehensive data model tailored for multi-dimensional event data, built upon the principles of LPG. The research in [55] focuses on creating a general data model designed to systematically store structural and temporal relations within a single, integrated graph-based data structure. This approach allows for a more comprehensive representation and querying of event data, especially when considering the multi-dimensional nature of such data. The model's design emphasizes the importance of capturing both the temporal sequence of events and the relationships between different events and entities in a unified manner. The proposed graph schema proposed in this paper is illustrated in Figure. 2.1. It comprises four distinct semantic node types: (i) logs, (ii) events, (iii) entities, and (iv) event classes. It also encompasses three structural semantic relations that establish connections between events, linking them to (i) one or more entities, (ii) exactly one log, and (iii) one or more event classes. Additionally, two behavioral semantic relations describe (i) the "directly-follows" relationship between two events (along a designated entity) and (ii) its counterpart "directly-follows" relation between event classes. This graph enables the representation of information at both the instance level and the model level.

In our research, we place a significant emphasis on the integration of both structural and behavioral data within a unified metamodel, akin to the approach delineated in [55]. However, our methodology diverges in two notable ways. First, our focus is solely on instance-level information, omitting any details related to the model level. Second, our metamodel features a

unique architectural framework, characterized by its distinctive node and relationship types. These choices are driven by our overarching objective to construct a highly expressive metamodel. This metamodel is carefully designed to encompass and explicitly represent every concept and relationship inherent to multi-dimensional process data, ensuring a comprehensive and insightful representation.

2.3 Natural Language Processing in Databases and Technical Domains

In this section, we provide an in-depth exploration of the NLP landscape within the domains of databases and technical fields. Specifically, Section 2.3.1 offers a comprehensive overview of the developments in constructing NLIs customized for interacting with database systems. Furthermore, in Section 2.3.2, we delve into the primary approaches related to intent detection and entity extraction, which constitute the fundamental NLP tasks employed in conversational systems.

2.3.1 Natural Language Interfaces to Database Systems

Several bodies of research have been proposed to assist users in querying database systems. These researches aimed to provide user-friendly interfaces for easily access the stored data. Existing works are classified into two categories: those that propose visual interfaces and those that propose textual interfaces. First category is related to visual query systems (VQS). These works propose a visual query language (VQL) [35], which employs a visual representation to depict the domain of interest. They also provide a visual language for expressing the query in a visual format. The authors in [101] compare existing VQS based on the visual representation used to depict the data model and to express the query. The VQLs are also used for querying different database formats: relational databases (e.g. [22]), object-oriented databases (e.g. [94]), data stream (e.g. [35]), web data XML (e.g. [36]) or RDF (e.g. [73]) and also graph databases (e.g. [75, 167]).

Second category is related to NLIs for querying databases (NLIDB) [29, 135, 148, 151, 177, 50] to which our work is closely relevant. NLIDB systems interpret a NL query, then translate it to a structured query to be executed over the database. Existing approaches can be categorized into rule-based (e.g. [29, 135, 148, 175]) or machine-learning based approaches (e.g. [151, 177, 50]). The authors of [3] review most recent rule-based approaches for constructing structured queries from NL utterances. They also provide a comparison of the techniques used and the type of questions that can be answered.

These approaches provide a high level of flexibility and adaptability as they are independent of any databases. The same set of patterns or rules are applicable to multiple use cases. The main limitation of rule-based methods is their limited scope since they make several assumptions about the database schema, the query language, and the NL queries. Indeed,

rules are insufficient to cover the wide range of NL, severely limiting the approaches' usability. Furthermore, all rules are handcrafted by developers, making it impossible to cover all cases. On the other hand, when the number of rules and their complexity increase, maintaining them becomes a major issue.

Recently, machine-learning based methods have been proposed [151, 177, 50]. In general, these approaches employ sequence to sequence models that translate NL queries to SQL queries [151, 177, 50]. The basic idea is to employ a machine learning model that takes the NL query as input and attempts to predict the corresponding SQL query. In other words, translating from NL to SQL can be expressed as a supervised machine learning problem involving pairs of NL and SQL queries. The main goal is then to predict the output sequence (i.e. SQL tokens) given an input sequence of tokens (i.e. NL tokens) based on previously observed patterns.

The main advantage of machine learning-based approaches over traditional NLIs is that they support linguistic diversity. However, one of the most significant challenges in developing these methods is the lack of training data [3]. Moreover, these systems act as a black box, preventing the user from knowing whether failures that may occur are due to linguistic issues in the query or because the database does not contain the desired result.

In this thesis, we propose a hybrid NLI system that combines rule and machine-learning-based approaches. First, we provide users with the flexibility to express their queries in NL (i.e. machine learning for NLP and NLU). Second, our approach does not require a large training dataset for constructing complex database queries from NL (i.e. rules-based for query construction).

The majority of current NLI systems are proposed for querying relational databases [29, 23, 135] or RDF graphs [142, 175]. Our work is related to [149] in which an approach is proposed to construct a structured subgraph from NL. However, this work is limited to the construction of a subgraph and does not address the problem of converting the subgraph into a graph query.

2.3.2 Approaches to Intent Detection and Entity Recognition

A variety of machine learning techniques and NLP models have been employed to perform intent recognition [31] and entity extraction [64]. Traditional approaches relied on rule-based systems and manually crafted features, often combined with classifiers like Support Vector Machines or Conditional Random Fields. While these methods offered reasonable results, they were limited by their reliance on handcrafted features and rules, making them less adaptable to complex language patterns. With the advent of deep learning, the NLP landscape underwent a significant transformation [97]. These models can be broadly categorized based on their approach: independent modeling and joint modeling. Independent modeling approaches treat intent classification and slot filling as separate tasks. For intent classification, methodologies range from Convolutional Neural Networks (CNN) [83, 173], to Long Short-Term Memory (LSTM) networks [127]. Attention mechanisms have also been integrated with CNNs [174]

and further expanded into hierarchical structures [164]. [99] introduced a novel adversarial multi-task learning approach for this purpose. On the other hand, slot-filling techniques have been diversified, with [160] employing CNNs, [165] utilizing deep LSTMs, and more intricate methods like the encoder-labeler deep LSTM [88].

Joint modeling approaches, as the name suggests, simultaneously address intent classification and slot filling. Notable methodologies in this category include the CNN-CRF model [163], the Recursive Neural Networks [68], and the attention-based BiRNN model [98]. A particularly innovative model in this category is the slot-gated attention-based model [62], which leverages intent context for enhanced slot filling. Collectively, these advancements underscore the rapid evolution and diversification of deep learning techniques in NLU. Furthermore, transformer-based architectures, like BERT model [45], revolutionized NLP [158]. These pre-trained language representations, learned from extensive amounts of unlabeled text, allowed transformers to capture contextual dependencies and achieve state-of-the-art performance in various NLP tasks, including intent recognition and entity extraction.

In addition to these traditional and deep learning approaches, the NLU family includes platforms like Wit.ai [56] and Dialogflow [63]. Wit.ai, acquired by Facebook, and Dialogflow (formerly API.AI), developed by Google, are NLP platforms that offer tools and APIs for developers to build applications and services with intent recognition and entity extraction capabilities. These platforms abstract the complexities of NLU, enabling developers to create language-based applications without building NLU models from scratch. They provide a user-friendly interface to train custom language understanding models and have been widely used to develop chatbots, virtual assistants, and other conversational interfaces.

NL interactions in conversations often possess layers of meaning, ambiguity, and intricate user intentions. In the context of NLP and chatbots, the term "complex intent" refers to a user's intention that involves multiple actions, conditions, or levels of comprehension. In contrast to simple intents, which have a direct mapping to a single action or response, complex intents necessitate a deeper understanding of context, multiple procedural steps, or a combination of simpler intents. Consider the following example: "I would like to schedule a flight to Paris next Friday, returning on the following Monday, and also secure a hotel reservation near the Eiffel Tower for those dates." In this scenario, the user's complex intent involves three distinct actions: (i) booking a flight to Paris on a specific date, (ii) arranging a return flight on a different date, and (iii) reserving a hotel near the Eiffel Tower for the trip's duration. While each of these actions could be considered a simple intent in isolation, when combined in a single statement, they form a complex intent.

Several advanced techniques have been proposed to recognize complex intents in NL [126, 57, 30]. The study conducted by [30] is dedicated to enhancing the recognition of complex intents in human-bot conversations by harnessing context-based knowledge. It posits that recognizing complex intents can be significantly improved by incorporating composite dialog patterns alongside fundamental intent characteristics. This research introduces an approach that combines established NLP and ML techniques for extracting NL features, including basic intents and dialog acts. In addition, it employs a rule-based approach that leverages these features, along with contextual knowledge derived from composite dialog patterns and other

metadata, to formulate rules for recognizing complex intents. This approach places particular emphasis on capturing complex intents that naturally arise during interactions with services, notably API methods.

2.3.3 Synthesis & Discussion

Our survey of NLIDB systems highlighted the evolution from rule-based to machine learning-based approaches. Rule-based methods (e.g. [29, 135, 148, 175]), although providing flexibility, were limited in scope, assuming certain database schema structures and query languages. Machine learning-based methods (e.g. [151, 177, 50]), on the other hand, introduced adaptability to diverse linguistic patterns but faced challenges related to data scarcity and interpretability. In response to these limitations, our work proposes a hybrid NLI system, integrating rule-based and machine-learning approaches (detailed in Chapter 4). This hybrid model retains the flexibility of NL queries while mitigating the need for extensive training data, offering a promising solution to the existing challenges.

Furthermore, in the realm of NLIs to databases, most existing methods concentrate on generating SQL or SPARQL queries. In contrast, our research is centered on constructing Cypher queries tailored for accessing LPGs. Additionally, prevailing approaches often rely on generalized datasets and benchmarks associated with specific domains. In contrast, our dataset is process-oriented, inherently distinct in its nature from the data utilized in these alternative methods.

2.4 Incorporating Cognitive Capabilities into Process Analysis

In this section, we delve into three distinct categories of works closely aligned with our contributions, all geared toward facilitating cognitive process analysis. Section 2.4.1 delves into the application of process analysis on unstructured data, while Section 2.4.2 elucidates the techniques enabling NL queries over process data and their integration with process mining tools. Lastly, Section 2.4.3 explores the potential of harnessing Large Language Models (LLMs) for conducting intricate process analysis tasks.

2.4.1 Process Analysis from Unstructured Data

Several efforts have delved deep into the realm of process analysis, broadening its scope to encompass not just structured but also unstructured data sources. These endeavors predominantly target the extraction of process data and knowledge from textual sources. For instance, the study in [60] introduced a methodology to automatically derive business process models from textual narratives. Similarly, [59] presented a prototype adept at semi-automatically detecting process model elements within NL texts, utilizing a plethora of data sources like

documents and reports. These identified elements subsequently serve as building blocks for process model generation.

Another area of research has concentrated on the utilization of process mining techniques with textual data in conjunction with structured event logs. For instance, [131] introduces a novel approach to analyzing business process execution complexity by combining both textual data and event logs. For textual data-based complexity, the study employs a set of linguistic features, adapting them from previous work. Machine learning techniques are then applied to predict complexity using these features. For log-based complexity, relevant metrics from the event log are used. The study then performs correlation analyses and identifies significant differences. Afterward, a correlation analysis of two complexities and an analysis of the significant differences in correlations are performed. [153] introduces a predictive process monitoring framework that integrates text mining with sequence classification techniques to manage both structured and unstructured event payloads.

Certain works have specifically explored the intricate landscape of messaging systems. [53] offers a comprehensive overview of the contemporary advancements in discerning business processes and activities from such systems. In this context, several works have been proposed to analyze business processes from emails. For instance, [52, 54, 91] focused on analyzing email systems to discover activities, actors and business data. Then, they used the discovered information to mine processes that are entirely or partially carried out via emails.

Similarly, the work [145] harnesses unstructured data, such as user comments or emails, to uncover the implicit context of processes. By employing information extraction and text clustering techniques, it provides a nuanced understanding of the process landscape. [21] delves into the intricacies of process extraction from text, shedding light on the existing limitations and future challenges. On the other hand, [80] employs an NLI to derive topics and process activities from customer service dialogues, representing them in a standardized format. Lastly, [136] introduces the C4PM method, a unique blend of agile principles, systems thinking, and NLP techniques, to analyze behavioral patterns in organizational semi-structured or unstructured data.

2.4.2 Natural Language Querying

Recently, new research [14, 70, 13] focused on using NL queries to answer process analysis questions, which are closely related to our work. The authors of [14] proposed a method for answering NL queries over process mining data. This research presents an innovative architecture for an NL conversational interface tailored to process mining tasks. This interface seamlessly translates user questions posed in NL into logical queries compatible with existing process mining tools. It introduces an abstract logical representation for process mining queries, designed to be both tool-agnostic and easily translatable into API calls for specific tools. The translation process involves several key steps. Initially, a series of natural language processing (NLP) techniques, including tokenization, part-of-speech analysis,

and entity recognition, are applied. These techniques identify general entities (e.g., people, locations), general entities relevant to process mining (e.g., cases, events, activities), and domain-specific entities corresponding to specific process domains (e.g., attribute values). Subsequently, the parsed NL query is mapped to predefined rules, resulting in the generation of a logical representation based on the identified rule. Finally, the logical representation is mapped to a real API call compatible with a chosen process mining tool. While this approach showcases promising capabilities, a notable limitation is the absence of a clear categorization for supported process mining queries. Additionally, the system relies entirely on a rule-based approach, which may restrict its applicability to predefined rules and potentially limit flexibility in handling a broader range of user queries.

An extended version of [14] is proposed in [13]. The authors enhance the abstract logical representation for process mining queries, as initially proposed in [14], along with the corresponding set of semantic rules. This extension enables the handling of queries related to process behavior and process mining analyses. Additionally, it introduces a taxonomy for NL questions pertaining to process mining. This taxonomy aids in categorizing process mining queries and provides a framework for structuring the evaluation. The approach supports handling NL queries such as *'What is the average execution time of the process?'*, *'How many conformance problems have been identified?'*, *'What is the most common flow of activities?'*, *'What is the average cost of the approval task?'*.

Another important work presented in [70]. It focuses on leveraging NL queries for process data retrieval. This research extends the capabilities of Athena [135], which relies on ontology-based NLIs for database querying. The authors present an automated approach for constructing an ontology from event data, enriching it with domain-specific terminology relevant to the business context. A key innovation lies in their development of a bootstrapping pipeline, which utilizes the process automation-derived ontology to automatically configure each component of Athena with domain-specific settings. This innovative approach simplifies the domain adaptation process and expedites the setup of Athena, making it more accessible and adaptable to diverse business contexts.

Similarly, [129] centered around the measurement of process performance indicators (PPIs) by utilizing both textual descriptions and event logs. It takes as input a textual description of a PPI and produces an output by evaluating this PPI against a provided event log. The approach unfolds through a sequence of four primary steps. In the initial step, the primary focus is on extracting pertinent entities from the textual description of the PPI. For this task, a fine-tuned BERT language model is employed. Moving on to the second step, the extracted entities are matched against the content of the event log, thereby initiating the establishment of a measurable PPI definition. In cases where essential information is omitted, the third step comes into play, employing a variety of heuristics to fill in these gaps and comprehensively complete the PPI definition. Finally, the fourth step utilizes the finalized definition to compute the desired PPI.

To the best of our knowledge, our work is the first to propose a hybrid method for querying event logs stored in graphical representations using NL. Moreover, in our work, we employ an intent detection step that increases the accuracy of answering NL queries over process data.

2.4.3 Leveraging Large Language Models for Process Analysis

Large Language Models (LLMs) have reached a remarkable level of proficiency, exemplified by recent advancements like GPT-4, enabling them to perform a wide range of tasks almost on par with human capabilities. These models hold significant potential for enhancing the analysis of business processes, offering the promise of an NL process querying approach, underpinned by the vast domain knowledge they have absorbed during training. The quality of information extracted from LLMs, however, hinges on the formulation of well-crafted prompts [87]. To tackle this challenge, prompt engineering has emerged as a powerful technique [34]. In the realm of LLMs, prompt engineering revolves around the strategic design and optimization of prompts, which act as input instructions directing the model's responses. The effectiveness of prompt engineering spans various domains, including chemistry assistance [176], human behavior simulation [77], job classification [38], and more.

Incorporating LLMs into complex tasks such as process analysis and process mining has spurred several innovative initiatives. A noteworthy case study by [26] explores the practical utilization of LLMs in process mining. This study introduces a framework that leverages abstractions, scenarios, and prompt definitions to guide LLMs in the analysis of process data. The prompts devised in this approach encompass crucial details: (i) a high-level description of the task, (ii) specifications regarding the desired output format, (iii) the textual source from which information should be extracted, and (iv) when necessary, a small set of input-output pairs as illustrative examples. The study effectively demonstrates how, through carefully tailored prompts and contextual information, an LLM can extract meaningful insights from event logs, identify process anomalies, and even propose optimization strategies.

Another technical report, [25], proposes the integration of LLMs into the realm of process mining. This paper offers diverse prompting strategies to mitigate information loss resulting from earlier abstractions. By transforming process mining artifacts, like event logs or PetriNet process models, into textual descriptions, these abstractions are presented to the LLM alongside the user's NL query. This approach can yield two distinct types of responses: direct answers to the initial questions or the formulation of hypotheses that can be validated through database queries against the original data. Similarly, the work by [74] delves into the potential of LLMs in enhancing conversational agents within the domain of process mining. Building on prior studies in NLP tailored for conversational agents, this approach capitalizes on LLMs to facilitate interactive and insightful conversations in the context of process mining. Additionally, [65] illustrates the prowess of LLMs in handling text-related BPM tasks. This study applies a specific LLM to three illustrative tasks: extracting imperative process models from textual descriptions, extracting declarative process models from textual descriptions, and evaluating the suitability of process tasks, as described in the text, for robotic process automation.

In our research, we demonstrate the utilization of LLMs to accomplish two primary objectives. Firstly, we employ them to automatically generate process-oriented NL queries, which are subsequently used for training an intent detection and entity extraction learning model. Secondly, we leverage LLMs to automatically construct Cypher queries from NL queries, en-

abling us to access process mining services. These NL queries are designed to articulate the user's needs when searching for specific process mining services. To ensure high-quality responses, we implement a prompt engineering process that aids in generating prompts that effectively guide the model toward producing the desired outputs.

2.4.4 Synthesis & Discussion

In the landscape of cognitive process analysis, prior research has made significant strides in areas such as process analysis from unstructured data, NL querying, and leveraging large language models. Efforts in these domains have explored diverse methodologies, from extracting process insights from unstructured textual data (e.g. [60, 59, 52, 54, 91]), to integrating NL querying techniques with process mining tools (e.g. [14, 70, 13, 129]), and leveraging the power of LLMs for nuanced process analysis tasks (e.g. [26, 25, 74, 65]). However, the innovation embedded in this thesis lies in the seamless fusion of these diverse approaches. Unlike existing methods, our research presents a comprehensive solution that not only interprets unstructured user queries but also harnesses the intelligence of LLMs to generate NL training data and to construct precise Cypher queries for process mining services. The novelty of our approach is amplified by meticulous prompt engineering, ensuring accurate and contextually relevant interactions. This integration of NL querying techniques with the cognitive capabilities of LLMs represents a paradigm shift, empowering users to navigate the complexities of process analysis effortlessly.

2.5 Discoverability and Accessibility of Process Mining Techniques

In this section, we provide an overview of previous research efforts dedicated to improving the discoverability (Section 2.5.1) and accessibility of process mining techniques through the provision of web services and API solutions (Section 2.5.2).

2.5.1 Discoverability of Process Mining Techniques

Numerous systematic reviews and surveys have been carried out to highlight the progressive developments in the realm of process mining [137, 134, 154]. These scholarly works offer a comparative analysis of various techniques from assorted perspectives, providing an overview of the accomplished work as well as identifying the existing gaps within the field. Additionally, they present an up-to-date account of how process mining has been applied in specific domains like healthcare [134], business management [170], etc.

Moreover, several recent surveys have shed light on specific sub-domains within process mining. For discovery techniques, the work of [9] offers a systematic review and comparative evaluation of automated process discovery methods, underpinned by an open-source

benchmark. The study by [147] outlines the latest developments in both declarative and hybrid process discovery techniques. [140] conducts a literature review of process discovery methodologies that leverage domain knowledge, defining a taxonomy that enables systematic classification and comparison of existing approaches.

In the realm of conformance-checking techniques, [51] provides a thorough review and classification of conformance-checking methodologies, comparing them across different dimensions. Turning to prediction techniques, [106] delivers an in-depth analysis and comparison of existing prediction methodologies, identifying key principles and offering a comprehensive overview. In addition, [47] presents a systematic review of prediction methodologies, offering a framework for their analysis, categorization, and comparison.

These studies serve as valuable resources for users and organizations seeking to identify process mining methods that meet their specific requirements. However, analysts still encounter challenges when it comes to accessing and implementing these methods in practice. Many process mining methods exist as standalone applications or are integrated into process mining tools like ProM [157], Disco [67], Apromore [90], etc. This can pose a hurdle for analysts as they may need to acquire specialized knowledge and familiarize themselves with different tools or applications to utilize the desired methods. Furthermore, certain methods are implemented within software tools that lack compatibility with other applications, limiting their adaptability.

2.5.2 Accessibility of Process Mining Techniques

To address accessibility and interoperability issues, the use of APIs and Web services in process mining has gained significant attention [92]. For instance, PM4Py [27] and bupaR [72] have incorporated process mining features such as log preprocessing, performance analysis, and visualization as libraries that can be integrated with the corresponding Python and R ecosystems. [92] presents a systematic introduction of Web services in the process mining field. Several commercial tools (e.g. Celonis[1] and Everflow[2]), as well as open-source projects (e.g. Apromore [90] and PM4Py-WS [28]), provide Web-based interfaces supported by Web services. Process mining analyses based on Web services enable easy integration with other software solutions. For instance, the business logic in Apromore is offered to the Web application through servlets-based Web services, providing external tools with the capability to utilize the algorithms integrated in Apromore through querying its Web services. Everflow, a platform for process mining, allows REST API access to a variety of techniques such as process discovery and bottleneck analysis. PM4Py-WS has developed Web services that are built on top of the process mining library PM4Py, which facilitated the integration of process mining techniques.

Despite the progress made in incorporating APIs and Web services into process mining, certain limitations persist. One such limitation is the incomplete coverage of process mining

[1]https://www.celonis.com/
[2]https://www.everflow.ai/

aspects in existing service-based solutions, with prediction being a notable gap. Another major limitation is the lack of thorough documentation that provides users with a clear understanding of the available services' properties, necessary inputs, and expected outputs. In addition, to the best of our knowledge, there is currently no solution available that aids analysts in finding a specific service and understanding its characteristics.

2.5.3 Synthesis & Discussion

In the realm of process mining research, previous endeavors have predominantly concentrated on enhancing either the discoverability (e.g. [137, 134, 154, 9, 147, 140, 51, 106, 47]) or the practical application of process mining techniques (e.g. [90, 28, 92, 72, 27]). Regarding discoverability, extensive efforts have been invested in conducting systematic reviews and surveys, illuminating diverse methodologies through comparative analyses. These resources play a crucial role in providing analysts with an overview of available process mining methods, aiding them in selecting the most suitable one tailored to their specific needs. Despite these advancements, analysts often face challenges in accessing and effectively implementing the identified methods.

On the other hand, approaches have been introduced to tackle the issue of accessibility and integration of process mining techniques. These approaches include API-based solutions and web service-based solutions. While these solutions aim to enhance the accessibility and integration of these methods, existing service-based solutions in the field of process mining have notable limitations. Particularly, they lack comprehensive coverage, especially in the realm of predictive analysis. Moreover, a significant drawback lies in the absence of detailed descriptions for available services, hindering analysts' ability to easily discern service properties, required inputs, and expected outputs. Furthermore, there is an absence of solutions aiding analysts in the automatic discovery and accessibility of process mining-related services.

In response to these challenges, our research introduces a pioneering service-based solution that overcomes these limitations (detailed in Chapter 5). Our innovative approach not only facilitates effortless access and integration of process mining methods but also empowers analysts by enabling the efficient discovery of discovery, conformance, and prediction services, aligning seamlessly with their requirements through the utilization of NL.

2.6 Web Service Description and Matching Approaches

In this section, our focus centers on the domain of service-oriented research. In Section 2.6.1, we delve into methods related to service description, while in Section 2.6.2, we explore approaches associated with the discovery of web services.

2.6.1 Service Description Approaches

Service description, a cornerstone in the realm of modern computing, serves as an intricate blueprint for various digital services, meticulously detailing their functionalities, operations, and access protocols. This comprehensive delineation ensures that both developers and end-users can effortlessly discover, integrate, and harness these services, eliminating the need to navigate the complexities of their underlying code. Within this domain, several service types have emerged, each with distinct features:

- **Web services:** emerge as pivotal bridges in the digital landscape, facilitating seamless communication between diverse software applications across myriad platforms and frameworks [6]. These services predominantly employ standardized protocols, such as SOAP [66], REST [162], and XML [115], ensuring smooth interoperability;
- **Cloud services:** these are services provided on-demand to users over the internet from the cloud computing provider's servers [138]. Their modular nature, segmented into IaaS [104], PaaS [166], and SaaS [155], ensures that businesses and individuals can tailor their cloud experience, scaling resources as per their evolving needs;
- **Semantic web services (SWS):** combine the principles of the Semantic Web and Web Services [110]. They use ontologies to make web services' functionalities machine-understandable, enabling automated service discovery, composition, and execution. Through semantic annotations and frameworks like OWL-S [107] and WSMO [49], SWS allows for a more efficient and automated interaction between machines on the web, reducing the need for manual intervention in service integration and execution;
- **IoT services:** refer to the suite of services designed to support and enhance the functionality of the Internet of Things (IoT). In essence, IoT services enable the seamless integration and management of smart devices, ensuring they work together efficiently and securely to deliver desired outcomes [169].

Existing methodologies for describing services in the realm of modern computing can be methodically categorized based on several essential dimensions. One pivotal dimension revolves around the type of model used in the description process, leading to two primary categories: syntactic models and semantic models. Syntactic models, as their name suggests, prioritize the structure and format of service descriptions (e.g. [37, 169]). A prominent example is the Web Services Description Language (WSDL) [37], which meticulously outlines the operations, messages, and interaction protocols of web services. In contrast, semantic models transcend mere syntax, delving deeper to capture the inherent meaning and context of services (e.g. [5, 102, 89, 171]). This profound understanding enables richer service discovery and composition. For instance, Web Service Modeling Ontology (WSMO) [81] provides a conceptual framework and a formal language for semantically describing various aspects of web services, including their capabilities, interfaces, and the underlying ontologies. By using WSMO, web services can be described in terms of standardized concepts and relationships, facilitating a deeper semantic understanding. Similarly, Web Ontology Language (OWL) [109]

describes service in terms of standardized concepts, properties, and relationships allowing for a deeper semantic understanding. The second dimension revolves around the intended purpose of the service description. Some methodologies are primarily designed to discover suitable services based on specific criteria (e.g. [111, 100, 114, 116, 82]), while others pivot toward service composition, focusing on the integration of multiple services to manifest particular functionalities (e.g. [89, 8, 12]). Additionally, certain approaches prioritize interoperability, ensuring that diverse services can seamlessly coexist and function in harmony, offering a smooth and unified user experience (e.g. [141, 117]). Lastly, the third dimension concerns the granularity of service descriptions. Within this dimension, distinctions can be made among atomic service descriptions, which detail singular, standalone services (e.g. [128]); composite service descriptions, which encapsulate conglomerates of interlinked services (e.g. [12]); and microservices descriptions, which spotlight smaller, modular services tailored for specific tasks (e.g. [144]). Each granularity level provides a unique perspective, catering to various needs and applications in the expansive domain of service-oriented architecture.

REST web services are inherently lightweight and stateless, and unlike SOAP-based web services, they don't have a standardized description language like WSDL. However, given the rise in the usage and importance of RESTful web services, several approaches have been proposed for their description, discovery, and composition. Categorizing the approaches for REST web services description can be done based on various factors. Some languages are designed specifically for describing the details of web services, typically including endpoints, methods, request/response formats, and so forth (e.g. [69, 150]). Other approaches aim to imbue web service descriptions with semantic context or hypermedia-driven details. For instance [143] provides semantic annotations for RESTful services integrating with SAWSDL. Similarly, [79] proposes a reference ontology for REST services along with a formal procedure for converting OpenAPI service descriptions to instances of this ontology.

2.6.2 Web Service Matching Approaches

Web service matching is a pivotal concept in the realm of web services, especially when it comes to catering to specific user requests. At its core, web service matching refers to the process of identifying and aligning web services that best fit a user's requirements or queries. This process is crucial for ensuring that users are provided with services that are most relevant to their needs, thereby enhancing the overall user experience. Among the myriad methodologies that have been developed to address this challenge, semantic web service matching has emerged as a particularly promising approach. Unlike traditional methods that rely primarily on textual or structural comparisons, semantic web service matching delves deeper. It harnesses the power of ontologies, semantic annotations, and comprehensive service descriptions to grasp the inherent meaning and relationships between different services. The primary objective of this approach is to ensure a more profound, context-aware matching process.

In the intricate domain of semantic web service matching, a myriad of methodologies have been proposed, each offering unique perspectives and techniques. These methodologies are categorized along two fundamental axes: the manner in which services are semantically

represented and the inherent functional behavior they exhibit. From a representational standpoint, three distinct methodologies have gained prominence. The first is the Ontology-based Matching approach. These methods leverage domain-specific ontologies to convey semantic descriptions of both service requests and available services. By grounding these descriptions in a standardized lexicon of concepts and relationships, this approach provides a profound semantic context, thereby facilitating the identification of matches predicated on ontological equivalences or similarities. It effectively aligns the matching process with a rich semantic landscape, where services are understood in the context of established ontological structures [107, 81, 4, 84, 93]. For instance, OWL-S [107] a semantic web service ontology that facilitates service matching and composition. It extends the OWL to describe key aspects of web services, including their inputs, outputs, preconditions, and effects. By annotating web services with OWL-S descriptions, developers and systems can discover and compose services that align with specific user requirements and contextual constraints. Similarly, WSDL-S [4] is an extension of WSDL that allows the semantic annotation of web services. These semantic annotations can be used for enhanced service matching and discovery. Furthermore, [93] explores the use of ontologies for personalized RESTful web service discovery.

The second methodology, Graph-based Matching, conceptualizes services and requests as multifaceted graph structures, such as RDF. The matching process revolves around comparing these graph structures, patterns, or topologies. This approach excels in scenarios where structural relationships and interconnections among service elements are critical to discerning meaningful matches. For instance, [5] proposes a model centered on hypermedia, enabling the generation of a graph that captures state transitions at the activity layer. It also captures resource, transition, and response semantics at the semantic layer. Using queries to traverse this graph facilitates discovery and composition. Furthermore, [172] introduces the Weighted Service Goal Model to enhance RESTful service discovery. By understanding the underlying goals or functionalities of services, this approach aims to provide a more accurate and meaningful discovery process. The third, Logic-based Matching, ventures into the realm of formal logic. It employs constructs such as description logic to identify matches through the establishment of semantic rules or axioms. By employing formal logic, this methodology brings a layer of deductive reasoning into the matching process, enabling a deeper understanding of service compatibility that transcends surface-level analysis (e.g. [125, 146]).

Conversely, the diversity among existing semantic web service matching approaches stems from their varying considerations regarding the aspects of services used for matching. This divergence extends beyond representation and centers on what services are described and leveraged in the context of matching. Some methodologies focus on the sequencing of operations or states exhibited by services. These approaches determine service similarity based on the likeness of the sequences of actions they undertake. For instance, two services might be deemed similar if they share a sequence of operations such as "login," "search," and "checkout." This sequence-centric approach accounts for the procedural nature of services and the flow of operations they execute, ensuring that matched services align in terms of their action sequences (e.g. [5]). In contrast, other methodologies prioritize the compatibility of inputs and outputs [84, 107]. These methods assess a service as a potential match if its output aligns seamlessly with the input requirements of a user's request or if its expected input mirrors the

output of another service within a broader workflow. This approach ensures the smooth integration of services into larger compositions, where data and information interchange smoothly between interconnected services. Additionally, a subset of approaches considers the intrinsic features of services. These features encompass a wide array of attributes, such as performance metrics, reliability, cost-effectiveness, and more. Such an approach enables users to select services that align with specific non-functional requirements, thereby enhancing the overall quality of service compositions and ensuring their adherence to desired criteria beyond mere functional compatibility.

Recognizing the complexity of service matching, there has been a surge in innovative approaches that harness the capabilities of NLP to simplify the service matching task. By integrating NLP, these methodologies aim to bridge the gap between technical terms and user-friendly language, making the process more intuitive and accessible. For instance, the approach proposed in [178] leverages the transformation of NL inputs into SPARQL queries, facilitating seamless interactions with semantic web applications without the need for users to understand the underlying query language. Similarly, [2] offers a novel method that employs NLP techniques to automatically match user requests, articulated in everyday language, with the most fitting semantic web service descriptions. By doing so, it eliminates the need for users to navigate the often convoluted technical descriptions, making service discovery more user-centric. Collectively, these approaches underscore the potential of NLP in revolutionizing the way we approach and understand service matching, making it more inclusive and less technically demanding.

2.6.3 Synthesis & Discussion

In the expansive field of web services description and web services matching, our research carves out a unique niche by focusing intently on process mining methods. While many existing methodologies offer generic solutions, our architecture is meticulously tailored to address the specific challenges and requirements inherent to the process mining domain. This specialization ensures a more precise and in-depth representation of services, capturing nuances that broader techniques might overlook. Moreover, while traditional graph-based matching techniques are prevalent, our approach innovates by introducing an LPG. This design not only encapsulates the properties, inputs, outputs, and micro functionalities of services but also establishes a technology-agnostic foundation, enhancing the flexibility and applicability of our service discovery process. Furthermore, our services matching component stands out by its ability to directly interpret user requirements, automating the creation of REST API calls. This user-centric approach, combined with our domain-specific focus, ensures that our matching process is both context-aware and tailored to the unique needs of process mining, setting our work distinctly apart from existing methodologies.

2.7 Conclusion

In this chapter, we have delved into various existing approaches relevant to the context of our thesis. Initially, we offered insights into works concerning process querying and process data storage, shedding light on existing research in the domain. Subsequently, we introduced works pertaining to the application of NLP in databases and technical domains. Furthermore, we explored studies that aim to incorporate cognitive capabilities to facilitate process analysis tasks. Additionally, we discussed research efforts aimed at enhancing the accessibility and discoverability of process mining techniques. Lastly, we explored existing methodologies for describing and discovering web services. These overviews served to not only highlight the existing body of work but also to identify the novel aspects of our thesis in comparison to previous research. In the forthcoming chapters, we will present our three key contributions. The next chapter presents our first contribution, which revolves around the introduction of a graph metamodel designed for the storage of process execution data.

Chapter 3

Graph Meta Model For Representing Process Execution Data

Contents

3.1 Introduction

In this chapter, we present the graph meta-model designed for representing multi-dimensional process data. This method directly addresses **RQ1-3** and is highlighted by the green pointed area in Figure 1.3. The main prerequisites guiding our meta-model's development are as follows:

- **Explicit Representation of Process Data:** The model should allow for the explicit representation of process data, treating the connections between data as a fundamental element. It must encompass all concepts and relationships relevant to multi-dimensional data.

- **Integration of Structural and Behavioral Information:** The meta-model should seamlessly integrate both structural and behavioral data into a unified graph.
- **Comprehensive Instance-Level Information:** It should comprehensively capture instance-level process execution details, including executed activities, involved actors, impacted data and objects, and the interconnections between these entities.
- **Compliance with Existing Standards:** The meta-model must comply with existing standards and storage techniques for storing and representing process data, ensuring that it can encompass all the concepts and relationships found in these conventional methods. Additionally, it should enable a richer representation of process data to facilitate comprehensive analysis and insights.
- **Efficient Querying:** The query language used to access the data should enable effective and efficient querying of process data, particularly for querying paths and complex connections.

To fulfill these requirements, we have proposed a graph meta-model based on Labeled Property Graphs (LPG) for storing and representing process execution data. LPG was chosen because it allows for an explicit and expressive representation of data connections. By defining the key concepts and relationships pertinent to single and multi-dimensional process data, we establish the minimum information required in the meta-model for expressiveness. We outline the primary graph elements, including node types, relation types, and properties, which are used to represent both structural and behavioral information. Additionally, LPG is queried using the Cypher language, providing a convenient and efficient means of querying paths within the graph.

We have qualitatively evaluated our approach for compliance with existing standards and its expressiveness. This evaluation was conducted using two publicly available event datasets. The first event data is related to the loan application process, and stored in CSV format. The second event data is related to the order management process, and represented in OCEL format.

The remainder of this chapter is organized as follows: We introduce the key concepts and relations related to single and multi-dimensional process data in Section 3.2. We provide motivation for the need for a graph meta-model for representing process execution data in Section 3.3. The proposed graph meta-model is detailed in Section 3.4, while the evaluation is presented in Section 3.5. Finally, Section 3.6 concludes the chapter.

3.2 Single & Multi-dimensional Process data

Information systems play a vital role in creating and modifying structured information records through transactions or activities during the execution of processes. These updates are captured as events, representing individual actions within the system. Events contain attributes

that describe the carried-out activity and the timestamp or other attributes indicating the order of executed activities. In the following sections, we will elaborate on the concepts and relationships pertaining to single and multi-dimensional process data.

Single-dimensional process data In a single-dimensional context, events are associated with a single entity identifier, allowing for correlation based on a specific entity and thus viewing the events from a particular perspective. In the following, we define the main process concepts and relations represented in a **single dimensional process data**:

C1 ***Event***: An event ev represents a specific activity or action that occurs within the process. It encompasses attributes such as the activity name ($ev.activity$), timestamp ($ev.timestamp$), and other relevant information related to the event execution.

C2 ***Resource***: Resources r refer to the individuals, systems, or entities responsible for performing the activities within the process. Each resource should be described by an identifier $r.identifier$ (e.g. name/email) and may have additional attributes such as the business/organizational role ($r.role$).

C3 ***Data***: Data represents the information associated with an event $ev.data$, including input parameters, output results, or any relevant data values that are manipulated or generated during the activity execution.

C4 ***case ID***: The case ID c is a unique identifier assigned to each process instance or case. It helps distinguish and link events belonging to the same process instance, allowing for the reconstruction and analysis of process traces.

C5 ***Trace***: A trace t represents the sequence of events that occur within a single process instance. It is identified by the shared case ID and provides a chronological view of the events associated with that case.

R1 ***Directly Follow Relation***: The directly follow relation describes the relationship between two consecutive events within a trace. It signifies that the second event immediately follows the first event in the process execution.

Consider an example of a customer support ticket management process. Table 3.1 illustrates an event log exemplifying concepts related to single-dimensional process data. Each row represents an individual event in the customer support ticket management process. In this process, each customer support interaction is captured as an event, representing individual actions within the system. For instance, an event (ev) in this context could be "Ticket Created," with attributes like the timestamp of when the ticket was submitted ($ev.timestamp$) and relevant data including the ticket ID, customer details, and the description of the issue ($ev.data$). Resources (r) in this system refer to the support agents responsible for handling customer inquiries. Each support agent is associated with specific events and is described by their unique identifier, such as their email or agent ID ($r.identifier$), along with additional attributes like their skill level or department. Upon a customer's submission of a support

Event ID	**Event Name** *ev*	**Event Timestamp** *ev.timestamp*	**Event Data** *ev.data*	**Resource** *r*	**Case ID** *c*
1	Ticket Created	2023-09-12 09:00:00	Ticket ID: 1001, Customer: Customer X, Issue: Technical Issue	-	Case 1
2	Assigned to Support Agent	2023-09-12 09:05:00	Agent ID: Agent A	Agent A	Case 1
3	Ticket Resolved	2023-09-12 10:30:00	Resolution: Issue resolved	Agent A	Case 1

Table 3.1: Example event log showing concepts related to single dimensional event data

ticket, the system assigns a unique case ID (c) to represent that particular ticket throughout its handling process. For instance, "Case 1" in Table 3.1, unfolds with the "Ticket Created" event, succeeded by the "Assigned to Support Agent" event, and ultimately concludes with the "Ticket Resolved" event. To maintain the chronological order of events within each case, the "Directly follow relation" is used. In the case of "Case 1", the "Ticket Created" event is immediately followed by the "Assigned to Support Agent" event, thereby signifying the step-by-step progression of the ticket management process. Thus, the trace t linked with Case 1 is: "Ticket Created" → "Assigned to Support Agent" → "Ticket Resolved."

Multi-dimensional process data In some cases, IS can include multiple distinct and uniquely identifiable entities. In such scenarios, the traditional notion of a case ID may not be present. Instead, data are grouped into objects and each event can be associated with one or more objects, providing the ability to view events from multiple perspectives simultaneously. Moreover, the objects themselves can have relationships with each other, allowing for correlations between events based on combinations of objects. In the following, we define the additional process concepts and relations related to **multi-dimensional process data**:

C6 ***Object***: An object o represents a distinct and identifiable element within the system. It is identified by a type (e.g. application, candidate, offer, etc.) and a set of data attributes that characterize it.

R2 ***Event-Object Relation***: This relation refers to the association between events and objects within the data. It represents the relationship that exists between a specific event and one or more objects to which the event is related or affected.

R3 ***Objects Relations***: Object relations capture the relationships and associations between different entities within the Information System.

R4 ***Directly Follow Relation Based on Objects***: The directly follow relation, in the context of multi-dimensional process data, considers the order and sequence of events based on the objects associated with them. It signifies that the occurrence of one event directly follows the occurrence of another event based on the objects involved.

Consider an example of a project management process. In this process, the system employs various objects to capture relevant information. For instance, an object o could be "Project",

"Task", or "Team". The "Project" object represents the overall project, each identified by a unique project ID, while the "Task" object signifies individual tasks with unique task IDs. Additionally, the "Team" object represents the teams responsible for executing the tasks, each identified by their team names. Events in the system are associated with these objects to provide a broader perspective. For instance, when a "Task Created" event occurs, it is linked to both the "Project" and "Task" objects, containing details such as the task ID, description, and estimated completion time. Similarly, a "Task Assigned" event is associated with both the "Task" and "Team" objects, indicating which team is responsible for a specific task. These event-object relations enable comprehensive analysis and allow tracking of project progress from various angles. Moreover, object relations, such as the correlation between "Project" and "Task," ensure effective organization of tasks within projects. Finally, to maintain the chronological order of events related to same objects, the "Directly follow relation based on objects" is used. For instance, for a particular project "Project 1", the "Task Created" event is directly followed by the "Task Assigned to Team" event, while for a specific task "Task 1" the "Task Created" event is directly followed by "Task Started" event.

3.3 Motivation

Effective storage and representation of process execution data are crucial for gaining valuable insights and optimizing business processes. Process execution data involves a multitude of entities and their intricate relationships (see Section 3.2). However, traditional storage techniques like XES event logs, relational databases, and OCEL face limitations in capturing the multi-dimensional nature of process data and efficiently querying complex relationships between entities. In the following, we will use the example of an order management process that is used to manage and fulfill customer orders. At the heart of this process lies the customer, whose needs initiate the entire sequence. Their request materializes in the form of an order, which can be placed through a myriad of channels, including online platforms, physical storefronts, or direct interactions with sales representatives. Once registered, the order undergoes a meticulous verification phase, ensuring both the availability of the requested items and the validity of the customer's payment method. Following this, the items are retrieved from inventory and consolidated into a package, ready for dispatch. The finance department, in parallel, manages invoicing and oversees the collection of payments, ensuring the transaction's fiscal integrity. The culmination of this process is the dispatch of the package, with dedicated logistics teams ensuring its timely and accurate delivery to the customer. Table 3.2 provides a sample event log associated with the order management process. This log encompasses key event attributes, including activity and timestamp, as well as the responsible resource. Additionally, it records pertinent objects, such as orders, customers, packages, and items. It's essential to acknowledge that there exist additional attributes linked to these objects. However, for simplicity, this example exclusively includes object identifiers, customer names, and email.

Traditionally, process data has been represented using XES event logs [71], where events are organized under a common case identifier (e.g., [124, 168]). As illustrated in Figure 3.1,

Activity	Timestamp	User	OrderID	CustomerID	CustomerName	CustomerEmail	PackageID	ItemID
Order Placed	|2023-09-01 09:00:00	Customer A	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM101
Order Placed	2023-09-01 09:05:00	Customer A	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Order Verified	2023-09-01 09:20:00	Sales Rep B	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM101
Order Verified	2023-09-01 09:20:00	Sales Rep B	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Payment Processed	2023-09-01 10:00:00	Finance Clerk C	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM101
Payment Processed	2023-09-02 12:00:00	Finance Clerk C	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Item Packed	2023-09-02 2:00:00	Warehouse Worker D	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Item Packed	2023-09-02 2:00:00	Warehouse Worker D	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Order Dispatched	2023-09-03 10:00:00	Logistics Team E	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM101
Order Dispatched	2023-09-03 10:00:00	Logistics Team E	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102
Order Delivered	2023-09-05 12:00:00	Delivery Person F	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM101
Order Delivered	2023-09-05 12:00:00	Delivery Person F	1001	CUST001	Customer A	customerA@email.com	PKG001	ITM102

Table 3.2: Event log for the order management process

this approach provides a representation of the event log from Table 3.2 using XML serialization. While effective for handling single-dimensional data, XES encounters challenges when it comes to capturing the intricate interactions among multiple entities within a process. In this context, events must be grouped under a predefined entity, often the order object as exemplified here, limiting the flexibility to explore the log from alternative perspectives. Moreover, the traditional XES event log structure lacks the capacity to represent the concept of objects, resulting in a significant loss of critical relationships among these objects. As a consequence, several essential components related to multi-dimensional process data, such as the "Object" representing entities like "Customer," "Order," "Package," and "Item," the "Event-Object Relation," the "Objects Relation," and the "Directly Follow Relation based on Objects," are not effectively captured within the XES representation.

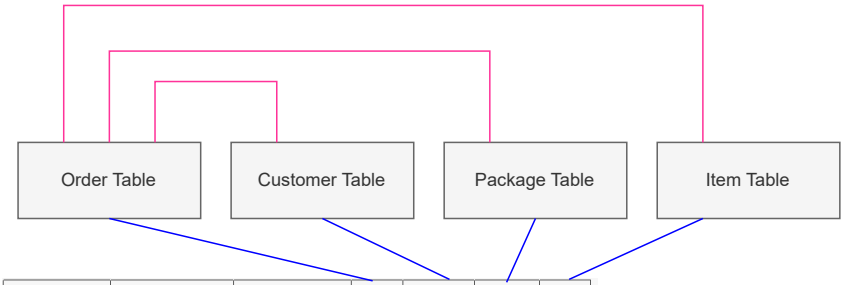
Figure 3.1: XES representation of the event log example associated with the order management process

On the other hand, the OCEL standard [61] provides improved semantics and event correlation compared to XES. It is a type of event log that organizes data based on distinct objects, rather than using a case-centric approach. Figure 3.2 provides an OCEL representation of the event log from Table 3.2 using XML serialization. While OCEL enhances event-object correlation (i.e. *"Event-Object Relation"*) and offers advantages in handling individual object interactions (i.e. *"Objects Relations"*), it may not fully address the complexities of capturing multi-dimensional process data and querying intricate relationships between entities in some scenarios. Querying complex relationships between entities in an OCEL representation can be challenging. As OCEL organizes events around individual objects, tracing and understanding the paths between related events across multiple objects (i.e. "Directly Follow Relation Based on Objects") may require additional effort and may not be as straightforward.

Figure 3.2: OCEL representation of the event log example associated with the order management process

Relational data modeling is a well-established technique employed to depict associations between events and objects, enabling the representation of both one-to-many and many-to-many relationships [139, 113], as exemplified in Figure 3.3. Nevertheless, this approach grapples with preserving the temporal sequence of events based on the involved objects. In Figure 3.3, while various relationships between events and objects, as well as among the objects themselves, are depicted, the explicit representation of the chronological order of executed events concerning specific objects remains absent. This deficiency poses a notable constraint, particularly when it comes to analyzing the order in which events transpire in

relation to individual objects. Moreover, the process of querying paths between events or objects within this framework presents notable challenges. The presence of multiple joins between tables, which is common in relational data modeling, considerably complicates the execution of intricate queries that demand the traversal of these relationships. Consequently, when dealing with complex queries, additional processing steps may become a necessity to extract the desired insights. This introduces an added layer of complexity to the overall analysis process.



Activity	Timestamp	User	OrderID	CustomerID	PackageID	ItemID
Order Placed	2023-09-01 09:00:00	Customer A	1001	CUST001	PKG001	ITM101
Order Placed	2023-09-01 09:05:00	Customer A	1001	CUST001	PKG001	ITM102
Order Verified	2023-09-01 09:20:00	Sales Rep B	1001	CUST001	PKG001	ITM101
Order Verified	2023-09-01 09:20:00	Sales Rep B	1001	CUST001	PKG001	ITM102
Payment Processed	2023-09-01 10:00:00	Finance Clerk C	1001	CUST001	PKG001	ITM101
Payment Processed	2023-09-02 12:00:00	Finance Clerk C	1001	CUST001	PKG001	ITM102
Item Packed	2023-09-02 2:00:00	Warehouse Worker D	1001	CUST001	PKG001	ITM102
Item Packed	2023-09-02 2:00:00	Warehouse Worker D	1001	CUST001	PKG001	ITM102
Order Dispatched	2023-09-03 10:00:00	Logistics Team E	1001	CUST001	PKG001	ITM101
Order Dispatched	2023-09-03 10:00:00	Logistics Team E	1001	CUST001	PKG001	ITM102

Figure 3.3: Relational data modeling of the event log example associated with the order management process

Graph-based data models such as RDF [130] and LPG [7] have been proposed as an alternative to overcome the limitations of relational modeling and querying. Graph data models are structured around data relationships, making it easier to represent and query complex relationships of multi-dimensional data. In the next section, we present the proposed graph metamodel based on LPG to model and query process execution data, inspired by the work of [55]. The metamodel allows for the modeling of different concepts and relations of multi-dimensional data defined in Section 3.2. In addition, the Cypher language[1] allows for easy access and query of the stored data.

3.4 Event property graph meta model & Cypher query groups

In this section, we introduce basic concepts related to LPG graphs and Cypher query language in Section 3.4.1. Next, we detail the proposed event property graph metamodel for

[1] https://neo4j.com/developer/cypher/

representing process execution data in Section 3.4.2. Finally, we present the Cypher query groups that we define to query different types of information from an LPG in Section 3.4.3.

3.4.1 Labeled Property Graph: LPG & Cypher Query Language

Labeled Property Graph: LPG An LPG is a directed labeled multigraph consisting of labeled nodes and relations. Nodes represent entities, and relations represent the relationship between two entities. Each node/relation may have properties that correspond to key-value pairs of attributes. A formal definition of an LPG is given in Definition 3.1. We assume that L is a set of labels (for nodes and relationships), P is a set of property names, V is a set of atomic values. For a set S, we denote by S^+ the set of all subsets of S excluding the empty set.

Definition 3.1 (**Labeled Property Graph**). *A labeled property graph is a tuple* $G = (N, R, \gamma, \lambda, \rho, \sigma)$ *where:*

- N *is a set of nodes;*
- R *is the set of relations;*
- $\gamma : R \mapsto N \times N$ *is a total function that associates each relation to a pair of nodes;*
- $\lambda : (N \cup R) \mapsto L^+$ *is a partial function that associates a node/relation with one or more labels from* L*;*
- $\rho : (N \cup R) \mapsto P^+$ *is a partial function that associates nodes/relations with properties;*
- $\sigma : (N \cup R) \times P \mapsto V$ *is a partial function that associates for each node/relation property a value from* V*.*

Figure 3.4 shows an example of an LPG that contains data related to e-commerce. It consists of nodes labeled with different categories such as *":Product", ":t-shirt", ":shoes", ":Order" and ":Customer"*, and each node has unique attributes as properties. For instance, the node in the top left corner represents a t-shirt product that has a brand of "Adidas", a color of "White", a size of "L", and a price of "$50". On the other hand, the node in the bottom left corner represents a customer named "Emma" with an address of "123 Main St.". Moreover, the edges in the LPG represent the connections between the nodes. For example, the customer "Emma" has placed an order with the number "O_654" that contains two products.

Cypher query language In an LPG, queries can be performed using a query language like Cypher[2]. Cypher is a widely used declarative graph query language. Each Cypher query is made up of (Clause, Patterns) pairs. A **Clause** specifies the type of operation to be used. A **Pattern** specifies the inputs that must be provided to these clauses. Below, we provide a comprehensive explanation of the five primary clauses used in a Cypher query: *MATCH-WHERE-WITH-ORDER BY-RETURN*.

[2]https://neo4j.com/developer/cypher/

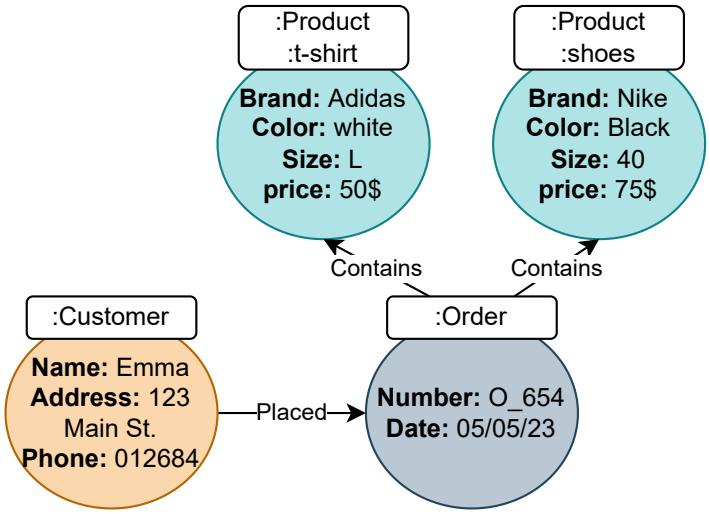


Figure 3.4: Example of LPG

- **MATCH clause**: enables the selection of sub-graphs with the same pre-defined pattern of nodes and relations.
- **WHERE clause:** restricts the selected sub-graph by adding conditions on nodes/relationship labels and properties.
- **WITH clause:** allows query parts to be chained together, piping the results from one to be used as starting points or criteria in the next. It is constructed after the WHERE clause and used to define new variables inside the query.
- **ORDER BY clause:** is a sub-clause specifying that the output should be sorted in either ascending (the default) or descending order. This clause normally follows the WITH or RETURN clause.
- **RETURN clause:** is used to define the output of the query which can be any graph elements (e.g. node/relationships or properties' values) as well as sub-graphs or any defined variables inside the query.

The example in Listing 3.1 demonstrates a Cypher query that consists of three clauses: MATCH, WHERE, and RETURN. The query aims to retrieve all products that a particular customer, namely Emma, has ordered. To achieve this, the MATCH clause identifies the path that links a customer to the order that includes the desired products. Then, the WHERE clause sets the condition for selecting the customer with the name Emma. Finally, the RETURN clause lists the matched products.

```
MATCH (c: Customer)-[:Placed]->(o: Order)-[:Contains]->(p:Product)
WHERE c.Name='Emma'
RETURN (p)
```

Listing 3.1: An example of a cypher query to get the products placed by certain customer

3.4.2 Event property graph meta model

LPG introduced in Section 3.4.1 enables flexible modeling of various concepts and their relationships [55]. Additionally, the Cypher query language facilitates easy access and querying of the stored data. In this section, we present a property graph metamodel that focuses on process event data modeling. It allows for representing the different concepts and relations of multi-dimensional process data defined in Section 3.2. Our proposed event property graph metamodel is presented in Figure 3.5.

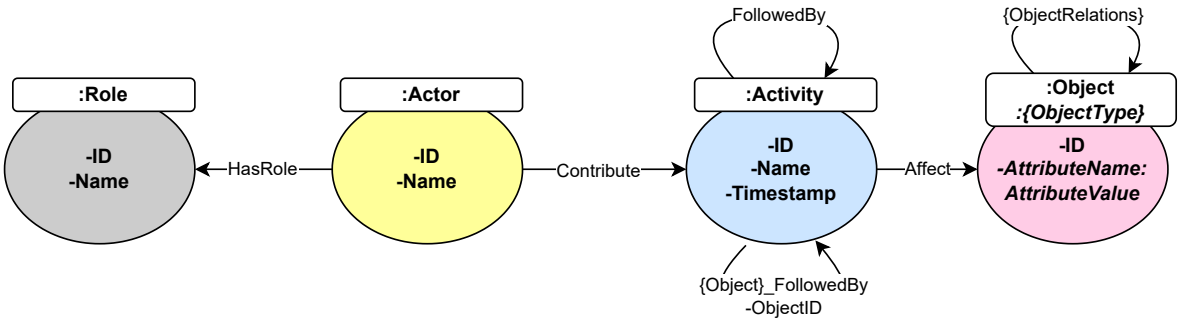


Figure 3.5: Event Property Graph Metamodel

The metamodel in Figure 3.5 describes the labels of nodes, the allowed relations, and the minimal set of properties required to represent event data. Nodes in the graph refer to the various types of information that can be extracted from event data. The *Activity* node refers to an activity instance executed within an event (i.e. concept *C1* defined in Section 3.2). The *Object* nodes refer to the artifact objects which group data manipulated by activities (i.e. concept *C6* and *C3* defined in Section 3.2). These nodes could have an additional label that represents the corresponding object type. The activity node has the activity instance id, name, and timestamp as mandatory properties. The Activity node is connected to the *Object* node through the relation *Affect* which indicates that the corresponding object is involved in the execution of the activity (i.e. relation *R2* defined in Section 3.2).

To model the temporal order between activity instances, we distinguish two types of relations: *FollowedBy* and *{Object}_FollowedBy*. The first relation allows to model the temporal order between all activity instances in the graph (i.e. relation *R1* defined in Section 3.2). The second allows to model the temporal order between activity instances from the perspective of a specific object (i.e. relation *R4* defined in Section 3.2). *Object* nodes may also be connected through the relation *ObjectRelation* which is replaced by the actual relation name (i.e. relation *R3* defined in Section 3.2).

In addition to *Activity* and *Object* node types, the *Actor* and *Role* nodes can be created in case information about the resources is available (i.e. concept *C2* defined in Section 3.2). The Actor node is connected to the Activity node through the relation *Contribute* which indicates the involvement of a specific actor in the execution of the activity. Similarly, the *Role* node indicates specific roles attributed to actors.

An example of an instantiation of our meta-model populated with loan application data is

illustrated in Figure 3.6. This example depicts seven executed activities, each with a unique name and execution time. The first three activities were carried out by User_37, while the last four were carried out by User_1. All activities affect the same Loan application which has an ID, an amount, a goal, and an application type as attributes. Furthermore, activities beginning with the letter 'O' affect the same offer node with the attributes ID, amount, monthly cost, and a number of terms. Moreover, two relations are added between the Offer and Application objects. These relations specify that the offer "Offer1" was offered and canceled in the application "App1".

Because these activities were carried out in the order specified by their timestamps, a FollowedBy relationship is added between them (arrows with blue color). Additionally, as they affect also the same application App1, an Application_FollowedBy relation is also added between them with the objectID attribute set to 'App1' (arrows with red color). Finally, for sequential activities affecting the offer Offer1, an Offer_FollowedBy relation with objectID attribute set to Offer1 is added between them (arrows with green color).

3.4.3 Cypher query groups

Cypher query language, introduced in Section 3.4.1, allows to easily access and query of the data stored in an LPG. In this section, we present the Cypher query groups that we defined to query different types of information. These groups will be used in the next chapter. We distinguish three groups of Cypher queries based on the type of the pattern selected in the MATCH clause:

- **Node matching queries** match a pattern with a single node. An example of such query on the event property graph in Figure 3.6 is MATCH (app:Application) RETURN app which selects all Application nodes and returns them.
- **Relationship matching queries** match a pattern that consists of nodes and relationships (i.e. a subgraph). Listing 3.2 shows an example. The query matches all subgraphs that consist of three node types Actor, Activity and Application connected through the relations Contribute and Affect.
- **Path matching queries**: is similar to relationship matching queries except that the relationships can be of variable length. For instance, the query MATCH (a1: Activity)-[FollowedBy*]-(a2: Activity) selects all subgraphs that consist of Activity node types connected to each other through the relation FollowedBy with a variable length (denoted with the '*' symbol). Therefore, the query matches all possible subsequences of activities.

```
MATCH (application: Application) - [:Affect] -> (activity: Activity),
      (activity) <- [:Contribute] - (actor: Actor)
WHERE activity.Name= 'A_Validated'
RETURN (actor),(application)
```

Listing 3.2: An example of a relation matching query

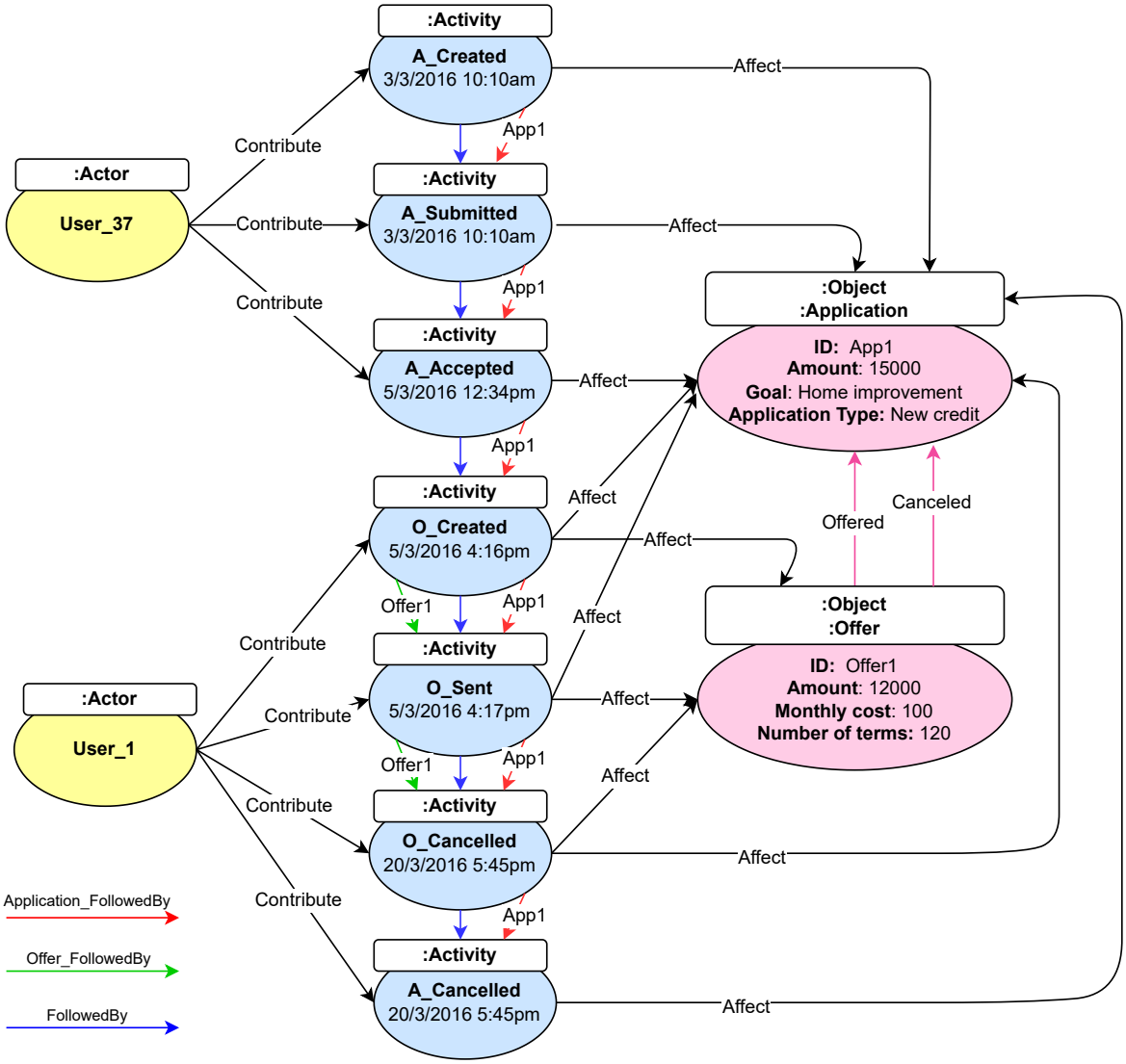


Figure 3.6: Event graph model populated with data related to loan application process

3.5 Evaluation & Discussion

This section presents the evaluation conducted to assess the effectiveness of the proposed metamodel. The main objectives of this evaluation are twofold: first, to evaluate the metamodel's compliance with existing standards and storage techniques, specifically by determining its ability to represent all the concepts and relations present in these traditional techniques. Secondly, we aim to assess the metamodel's expressiveness, specifically its capacity to represent additional information that might not be explicitly represented in traditional techniques. By doing so, we aimed to demonstrate the metamodel's capacity to provide a richer representation of process data, enabling more comprehensive analysis and insights.

To achieve these objectives we used real and public process data which are described in

Dataset	Format	Event records	Distinct activity names	Distinct object types	Object Instances	Actors
BPIC'2017	CSV	2105	26	3	Application: 154 Offer: 183 Workflow: 240	69
Order Management log	OCEL	22367	11	5	Product: 20 Customer: 17 Item: 8159 Offer: 2000 Package: 1326	-

Table 3.3: Datasets characteristics

Section 3.5.1. The methodology employed for the evaluation is detailed in Section 3.5.2, while the results are discussed in Section 3.5.3.

3.5.1 Process execution data

We evaluated the effectiveness of our approach using two publicly available event logs, each represented in a distinct format - CSV and OCEL. These event logs cover different process domains, providing diverse real-world data for our evaluation. Table 3.3 summarizes the characteristics of these event logs.

The first event log, BPIC'17 event log [123], captures the loan application process, encompassing the entire journey from application submission to the final decision-making (approval or decline). The log is stored in CSV files, and to streamline the evaluation, we filtered it by removing infrequent events with a frequency of less than 20%. Consequently, we obtained a dataset comprising 2105 unique event records, covering 26 different activities involving three distinct object types: application, offer, and workflow. Within the dataset, there are 577 object instances, including 154 applications, 183 offers, and 240 workflows. The process activities are executed by 69 different actors, and no business roles are included in this context.

The second event log is the order management log[3] , which is represented using the OCEL format [61]. This log focuses on customer order processing, monitoring orders from their acceptance to the final delivery. It comprises a larger dataset, containing 22367 unique event records, involving 11 different activities and five diverse object types: product, customer, item, order, and package. Within the dataset, there are 11522 object instances, including 20 products, 17 customers, 8159 items, 2000 orders, and 1326 packages. There are no actors or business roles included in this log.

[3]OCEL Standard: http://ocel-standard.org/

3.5.2 Methodology

The initial phase of our evaluation involved a thorough examination of the structure of the two datasets utilized. In this step, we familiarized ourselves with the datasets' structures, entities, relationships, and properties. Our aim was to identify the main concepts and relations explicitly represented in these datasets. To evaluate compliance, we meticulously mapped the entities, relationships, and properties from the datasets to their corresponding counterparts in our meta-model. This mapping process allows us to determine how effectively our meta-model accommodates the essential aspects found in these standard datasets.

Furthermore, to assess the expressiveness of our metamodel, we identified the additional concepts and relations that were necessary beyond what the existing standards provided. This supplementary information was explicitly incorporated into our meta-model, enhancing its capacity to represent complex multi-dimensional process data comprehensively. For instance, we delved into the specifics of each dataset to identify relationships between objects based on the underlying logic. In the loan application process, if an activity "O_Canceled" was related to both an application and an offer object, we added a "canceled" relation between the respective application and offer objects. Similarly, we derived and explicitly added the directly follow relations based on objects, harnessing the stored information to construct a more comprehensive representation. Lastly, we employed a data transformation script to automate the process of populating the graph in Neo4j. This script efficiently handled the data files as input and facilitated the seamless creation of the graph representation. The result of this transformation was the successful storage of the processed data as a graph database in Neo4j. For the BPIC'17 event logs, this yielded a graph comprising 2157 nodes and 10692 relations, while the order management log produced a graph with 33889 nodes and 750710 relations.

3.5.3 Results

After conducting the evaluation using the two publicly available event logs, we obtained valuable results that shed light on the effectiveness of our proposed meta-model to represent multi-dimensional process data.

Regarding the BPIC'17 event log, which captures the loan application process, our meta-model demonstrated a high level of compliance. Through meticulous analysis, we successfully identified and mapped the main concepts and relations present in the event log to their corresponding representations in our meta-model. Notably, the event log explicitly contained the following concepts: *Event*, *Data*, and *Object*, along with the *Event-Object Relation*. Similarly, for the order management process stored in the OCEL format, we achieved successful identification and mapping of the main concepts and relations to our meta-model. This event log also explicitly included the following concepts: *Event*, *Data*, and *Object*, as well as the *Event-Object Relation*. The seamless integration of these entities, relationships, and properties from the event log into our meta-model showcased its capability to accommodate essential aspects from existing standards and traditional storage techniques. The mapping process affirmed

our meta-model's effectiveness in handling diverse event logs while maintaining compliance with existing standards.

The evaluation further highlighted the expressiveness of our meta-model. By discerning and representing additional concepts and relations beyond what the standard datasets offered, we empowered our meta-model to capture intricate relationships between various object types and activities. Notably, we explicitly represented additional relations, such as the relationships between objects and the *Directly Follow Relation* based on both timestamps and objects. These enhancements showcased the adaptability and versatility of our meta-model in representing complex multi-dimensional process data.

Overall, our evaluation results demonstrate the substantial potential of our proposed meta-model in effectively representing and analyzing multi-dimensional process execution data. The high compliance with existing standards and the augmented expressiveness in handling complex relationships validate the utility of our meta-model as a valuable tool for process data representation and analysis in real-world scenarios.

However, a notable limitation is that for each new dataset, a new script must be implemented to populate the graph meta-model. This is mainly due to the inherent differences in data representations, particularly when dealing with data stored in CSV files. Additionally, the identification of additional concepts and relations for each dataset was a manual process, which hindered the generalization of the data transformation process. For instance, the naming of object relations needed manual identification based on the specific characteristics of each process.

3.6 Conclusion

In this chapter, we successfully accomplished **Objective 1.3**, which involved the selection and proposal of a suitable storage representation for process data. This choice played a pivotal role in the well-designed automation of structured query construction within our solution. This contribution directly addresses the sub-question (**RQ1-3**: What storage technique and query language should serve as the foundation for the automated solution to ensure both effective and efficient querying?) in our pursuit of answering the primary research problem (**RQ1:**) for making process data easily accessible for users in a natural manner.

To this end, we have introduced our graph meta-model designed to accommodate and explicitly represent every concept and relationship inherent to multi-dimensional process data. With a keen focus on instance-level details, it comprehensively captures information such as executed activities, involved actors, impacted data and objects, and the interconnections between these entities. Notably, we have placed considerable emphasis on the integration of both structural and behavioral data, unifying them within this metamodel. To ascertain its effectiveness, we have conducted an evaluation that measures the metamodel's compliance with existing standards and assesses its expressiveness, particularly its ability to capture additional information that might be overlooked by traditional techniques. This evaluation employed

two publicly available process datasets. It is worth noting that we have refrained from performing a comparative evaluation of our LPG model against other storage techniques, as this aspect has already been comprehensively examined by [55], establishing the effectiveness of LPG for process data storage. In the next chapter, we introduce our automated solution utilizing NLI, meticulously designed to automate the query process over data represented within our proposed LPG model.

Chapter 4

Natural Language Interface for Querying Process Execution Data

Contents

4.1 Introduction

Process data querying is a pivotal technique in the realm of process analysis. It enables analysts to pose granular questions, providing immediate insights into various aspects of process execution. For instance, questions like "Who validated job application 10?" or "When is the interview scheduled for Candidate C1?" exemplify the type of inquiries that process data querying can address. However, existing process query languages and technologies primarily target data scientists, assuming a certain level of technical expertise in using query languages like SQL, Cypher, or SPARQL, or dealing with low-level APIs.

This limitation is significant because it excludes domain analysts, including those in fields like healthcare or insurance, and even end-users, who should have the ability to leverage process analytics for their tasks within digitally enabled processes. Moreover, a key challenge with existing process analytics technologies is that they often do not make process data easily accessible to human users in a natural and user-friendly manner.

To address this limitation, there is a need for a solution that incorporates cognitive capabilities to facilitate process querying task. By Leveraging AI techniques, such as NLP and LLMs, process data should be accessible to users with varying levels of expertise. To this end, we present in this chapter a NL-based pipeline designed to assist end-users in querying process data. The user-facing interface takes a query formulated in NL and automatically generates a corresponding structured query to be executed over structured process data, and returning the response (highlighted by the blue pointed area in Figure1.3). We use the graph metamodel introduced in Chapter 3 for storing process data, and we aim to answer NL queries by automatically constructing the corresponding Cypher queries.

We categorize NL queries related to process data into three categories based on state-of-the-art process querying [121] and process mining [156] techniques: content, behavioral, and performance queries. This categorization covers a broad spectrum of aspects within process data, allowing users to explore content, behavior, and performance metrics.

Our proposed pipeline is hybrid combining machine learning and rule-based approaches. The pipeline comprises two main stages. In the first stage, we employ machine learning techniques for intent detection and entity extraction, using a model trained on a collection of NL queries paired with their respective intents and entities. To generalize the solution, we propose general patterns for intents and entities based on the LPG metamodel, which are then instantiated according to the event property graph model. In the second stage, a rule-based approach is employed to construct the corresponding database query based on the intent and entities identified in the first stage.

Furthermore, to minimize the effort required to adapt the system to multiple domains, we propose an automated approach that leverages prompt engineering and LLMs for the generation of NL queries used for training the machine learning model. By harnessing the capabilities of LLMs like GPT-4, we can automatically generate a diverse range of NL queries, eliminating the need for manually annotated datasets for each domain transition.

To evaluate our approach, we conducted experiments using two publicly available process datasets: one related to a loan application process and the other related to order management. We collected over 370 NL queries from external users who were unfamiliar with the Cypher query language, as well as 150 NL queries generated using a paraphrasing tool. These queries can serve as benchmarks for future research. Additionally, we conducted a comparative evaluation between the machine learning model and the rule-based approach for intent detection in NL process data queries.

Our research findings have undergone rigorous peer review and have been accepted for publication at the ICPM 2021 conference [85]. Furthermore, an extended and more comprehensive version has been published in the Information Systems Journal [86].

The remainder of this chapter is organized as follows: Section 4.2 provides definitions of relevant concepts related to intent recognition and entity extraction techniques. Section 4.3 presents motivation examples and discusses key challenges in automating the construction of graph queries. Section 4.4 provides an overview of our proposed approach, while Section 4.5 defines the NL query categories for process data. The two main components of the pipeline are elaborated in Section 4.6 and Section 4.7. Section 4.8 delves into our automated solution for generating NL training data related to the process domain using prompt engineering and GPT-4. Section 4.9 presents and discusses experiments and evaluation results. Section 4.10 concludes the chapter.

4.2 Backgrounds

Natural Language Understanding (NLU) is a pivotal domain in both research and practical applications within the broader field of NLP. Its primary objective is to bridge the gap between human language and machine comprehension, enabling computers to interpret and understand human language in a meaningful way. NLU plays a critical role in various applications, such as virtual assistants, chatbots, sentiment analysis, customer support systems, and more, where accurate language understanding is paramount to providing relevant and contextually appropriate responses.

In this section, we will focus on two fundamental tasks in NLU, which we will apply in our approach: Intent Recognition (presented in Section 4.2.1) and Entity Extraction (presented in Section 4.2.2).

4.2.1 Intent Recognition

Intent Recognition, also known as Intent Classification or Intent Detection, involves deciphering the underlying intention or purpose behind a user's input. When users interact with NLU-powered applications, they typically express their queries or requests in NL. The objective of intent recognition is to understand the user's intention and map it to a predefined intent category, representing the specific action or goal the user aims to achieve. For example, given

the user query, *"Book a flight from New York to Los Angeles on Friday"*, the intent recognition system would identify the user's intent as "Book Flight," enabling the NLU application to proceed with the flight booking process. Accurate intent recognition is crucial for designing efficient conversational systems as it allows for routing user queries to the appropriate functionality or service, significantly impacting the overall user experience. Misinterpreting the user's intent could lead to frustrating interactions and suboptimal responses.

4.2.2 Entity Extraction

Entity Extraction, also known as Named Entity Recognition (NER), involves identifying and categorizing specific entities or key information from the user's input. Entities can include names of people, places, organizations, dates, times, numerical values, and other domain-specific entities. Continuing with the previous user's query example *"Book a flight from New York to Los Angeles on Friday"*, entity extraction would identify entities such as "New York" (origin), "Los Angeles" (destination), and "Friday" (date). Extracting these entities from the user query is essential for understanding the context and obtaining relevant information required to fulfill the user's request accurately. Entity extraction is particularly valuable when dealing with unstructured text, such as user-generated content, social media posts, or customer support emails. By identifying and categorizing entities, NLU systems can organize and analyze information more effectively, leading to improved decision-making and streamlined information retrieval.

4.3 Motivation & Challenges

In this section, we motivate the automation of constructing the Cypher query from NL (Section 4.3.1). In addition, we will discuss several challenges that come with such automation (Section 4.3.2).

4.3.1 Motivation

To construct a Cypher query, the user should be familiar with (i) the Cypher language, and (ii) the graph structure where the data is stored. First, the user should learn about the various clauses that comprise a Cypher query (as detailed in section 3.4.1) as well as how to use these clauses to construct a query.

```
MATCH (application: Application) - [:Affect] - (activity: Activity),
      (activity) - [:Affect] - (offer),
      (offer:Offer) - [:Offered] - (application)
WHERE application.Goal= 'Car' AND activity.Name= 'A_Validated' AND
      activity.Time>='2021-12-01' AND activity.Time<='2022-03-01'
WITH  application, COUNT(offer) as offerCT
WHERE  offerCT>=3
```

```
RETURN (application)
ORDER BY application.Amount
```

Listing 4.1: An example of a cypher query for querying process data

Listing 4.1 shows an example of a Cypher query to answer the following NL query: *'Return all car loan applications validated within the last three months, and which received 3 offers minimum, order them by their requested amounts'*. In the following, we will refer to this NL query by **(NLQ1)**. The Cypher query selects all subgraphs in the event property graph in Figure 3.6 that consist of three node types Application,Offer and Activity. The node types Application and Offer should be connected through the relation Offered. The node type Activity is connected to Offer and to Application node types through the relation Affect. The selected subgraphs are conditioned in the WHERE clause which keeps only the subgraphs whose node properties satisfy the conditions. Afterwards, the WITH clause takes the selected subgraphs and counts the number of offers per application (akin to group by clause in SQL). The second WHERE clause applies a second filter and keeps the subgraphs whose application node is connected to at least three offers. Finally, the query returns the application nodes ordered by their requested amount property.

Second, the user should be familiar with the graph structure (i.e. node types and relations required to construct a valid sub-graph) and the different graph elements (i.e. exact property names and values to add correct conditions). For instance, the user should be aware that application validation is referred to by the activity name 'A_Validated'. These requirements, however, make the querying task inaccessible to business users who may lack such technical background. Thus, there is a need for an automated solution that keeps users away from these technical details.

4.3.2 Challenges

In this section, we discuss the main challenges that arise when processing a NL query to automatically generate the corresponding graph query.

Chal_1: Natural Language Variation. Users may express their intentions in NL in a variety of ways. For example, to retrieve the number of applications with an amount greater than 10,000, a user can enter queries such as *["what is the number of applications with an amount more than 10,000?", "How many applications are there with amount 10,000 at least?", "Count the applications with a minimum amount of 10,000.", etc.]*. Furthermore, each graph element can be expressed in a variety of NL expressions (e.g. actor could be expressed as person, user, resource, etc.). The same applies for activity names. For instance, the activity name 'A_Validated' is not stated explicitly in **(NLQ1)**, but rather was inferred from the verb 'validated' in the question.

Chal_2: Graph Elements Inference. Nodes types or properties names may not be explicitly mentioned in the text. The system should be able to infer automatically the node types from the node's property names or values, as well as property names from property

values that appear in the NL query. In **(NLQ1)**, for instance, the system should be able to recognize that the detected entity 'car' corresponds to a value of the property 'Goal' of the node type 'Application'. The same applies for the date values. For instance, the last three months expression in **(NLQ1)**, refers to a date period value of the property 'Time' of the node type 'Activity'.

Chal_3: Numerical Conditions Identification. Numerical values should be assigned to their corresponding properties with the appropriate operator. For instance, in the question *'Which offers have a minimum monthly cost of 200 and an offer amount greater than 15,000?'*, two conditions with numerical values should be correctly applied. The first condition is that the monthly cost property is $>= 200$. The second is that the offer amount is > 15000. Furthermore, the system should recognizes the conditions over the count variables. For instance, in **(NLQ1)** the system should recognize the condition that the number of offers is greater than or equal to three.

Chal_4: Syntactically Correct Sub-graph Construction. The query sub-graph constructed from the retrieved entities should be syntactically correct. A sub-graph is syntactically correct if it is compatible with the graph model in which the data is stored. However, in some cases, not all graph elements required to complete the sub-graph are retrieved from the NL query. For instance, in **(NLQ1)**, only activity, offer and application nodes are detected. These elements, however, are insufficient, and the two relations 'Affect' and 'Offered' should be added to complete the sub graph as shown in the MATCH clause of Listing 3.1.

In the following sections, we describe how we overcame these challenges to automatically construct the Cypher query, without making any assumptions about the NL questions.

4.4 Approach Overview

The pipeline developed in our approach is depicted in Figure 4.1. It takes an NL query as input, translates it into a Cypher query, and executes it on an event property graph to obtain the query result, which is then returned to the user. This approach is highlighted by the blue pointed area in Figure 1.3. We defined three main categories of NL queries associated with process data queries (presented in Section 4.5). The pipeline comprises two main components: the NLU component (elaborated in Section 4.6) and the query construction component (explained in Section 4.7).

NLU component: This component receives the user's query and performs NLU tasks. It allows the detection of the user's intent (i.e. what the user is asking about) and the extraction of named entities from the query (i.e. terms that correspond to elements in the event property graph). In this work, we adopt a machine learning-based approach for intent recognition and entity extraction since it is more robust against variations in NL.

In machine learning, intent recognition and entity extraction can be seen as a classification task in which a model is trained with a set of queries and their associated intents and entities.

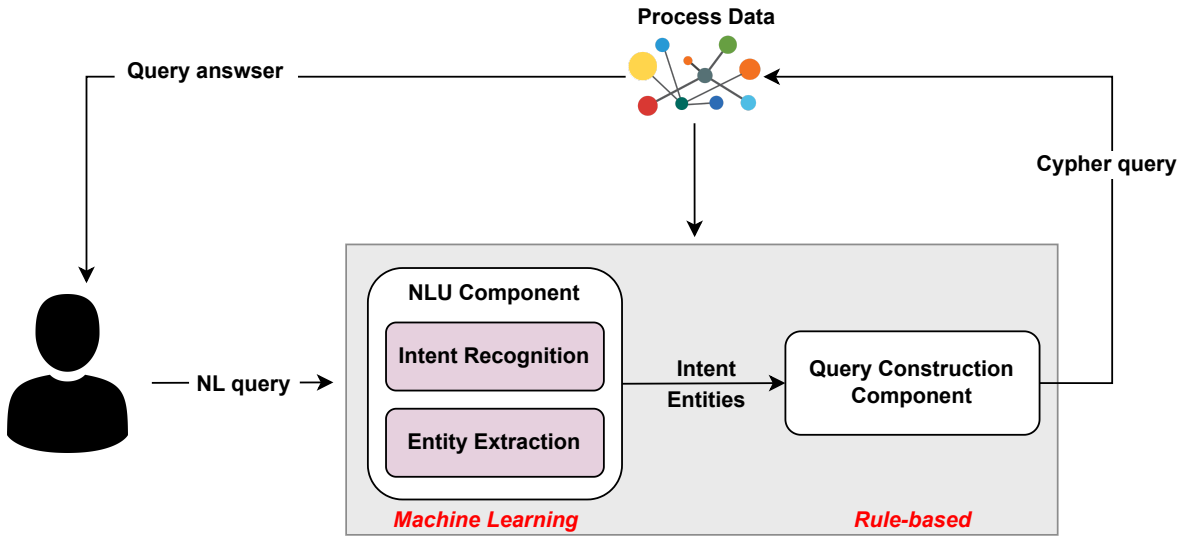


Figure 4.1: Overview of the proposed approach

Therefore, the main contribution of this component is the identification of the possible intents and entities that the system should learn to detect. In our work, as we aim to query process data, intents and entities should pertain to elements in our event property graph. Intents describe the type of information to be searched for and returned to the user, such as nodes, relationships, paths, node/relationship properties, or any aggregation applied to numeric properties. Entities refer to graph elements (e.g., node types, node properties, relation types). We propose general patterns that are defined based on the LPG meta-model in general and that are instantiated according to the event property graph model. Once defined, we can apply state-of-the-art classification techniques for intent recognition and entity extraction.

Query Construction Component: The query construction component takes the detected intent and entities from the NLU component as input and automatically constructs the corresponding Cypher query. It is a rule-based component consisting of four main steps. First, the detected intent determines the Cypher query group (node, relationship, or path matching query) and the minimal elements required in the MATCH clause. It also specifies the elements of the RETURN clause. Secondly, the detected entities are added either as conditions in the WHERE clause or as elements in the subgraph of the MATCH clause. Entities corresponding to nodes and relationships are added as subgraph elements, while those corresponding to values are added as conditions. Third, the WITH or ORDER BY clauses are added if there is a need for aggregation or if the user requires the results to be ordered. This is determined by analyzing the presence of trigger words (e.g., "in each," "for each", etc.) that indicate the need for aggregation or words (e.g., "ordered", "sorted", etc.) that indicate the need for ordering the results. The algorithm examines the grammatical relationships, provided by the dependency parser, between the trigger words and each entity in the NL query to determine based on which entity the results should be aggregated or ordered. Finally, the Cypher query is completed to be syntactically correct.

After constructing the Cypher query, it is executed on the stored graph, and the response is returned to the user. In the following sections, we detailed the two components in our pipeline.

4.5 Process Data NL Queries Categories

We divided process data NL queries into three main categories based on state-of-the-art process querying [121] and process mining [156] techniques: content, behavioral, and performance:

- **Content queries:** Content queries aim to provide answers to questions regarding events within a process and the associated data. They furnish information about the specifics of one or multiple events. For instance, content queries can address inquiries such as:
  - Executed activities (e.g. "What activities were carried out last month?");
  - Involved actors (e.g. "Who is responsible for submitting loan applications?");
  - Data objects (e.g. "Which offers with an amount less than 10,000 were rejected?");

  Content queries essentially delve into the tangible aspects of the process by focusing on events and their attributes.
- **Behavioral queries:** Behavioral queries are designed to facilitate the exploration of process execution behavior. They enable users to inquire about various aspects of how activities are carried out. For instance, behavioral queries can address questions like:
  - Temporal execution order (e.g. "Which activity follows the submission of an application?", "Give me the sequence of activities that occur after the validation of App1");
  - Process instance traces (e.g. "Provide me with the trace of activities related to the processing of App1");

  Behavioral queries offer insights into the dynamic flow and sequence of activities within a process, allowing users to understand the order and relations between them.
- **Performance queries:** Performance queries aim to provide answers to questions related to the efficiency and timing aspects of a process. They encompass inquiries about various performance metrics, such as:
  - Processing time (e.g. "How long does it take for App1 to be processed?");
  - Delay analysis (e.g. "What factors contribute to delays in processing loan applications?");

  Performance queries focus on assessing the effectiveness and efficiency of a process, allowing users to identify areas for improvement and optimization.

These three query categories collectively cover a wide range of aspects within process data, offering users the ability to explore content, behavior, and performance metrics. Each of these categories requires dedicated intent definition and Cypher query construction mechanisms.

4.6 NLU Component

This component analyzes the user's NL query by performing two main NLU tasks: intent detection and entity extraction. It aims to address **Chal_1** and **Chal_2** challenges described in Section 4.3.2, by defining the possible intent and entity classes that could be associated with the NL queries. Figure 4.2 shows a simple example of the detected intent and extracted entities from a NL query related to the event property graph in Figure 3.6. The system

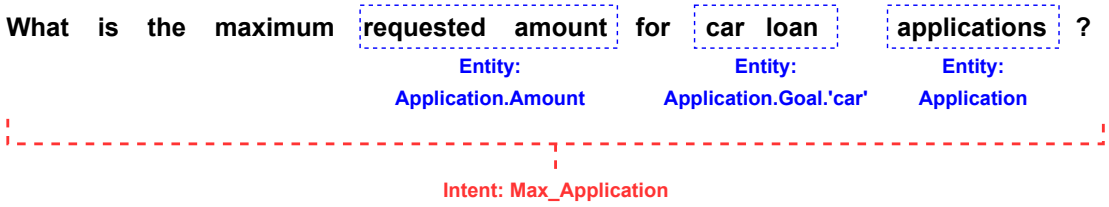


Figure 4.2: Example of detected intent and extracted entities from an NL query

detects that the question intent is Application_Affect which means that the user is inquiring about the node Application affected by some activity node. The system also extracts some entities such as application, validated, and John. This example shows that intent and entities are clearly domain-specific, limiting the pipeline's applicability. To address this problem, we defined general patterns for intent and entities that are easily instantiated according to the specific data model. Patterns are defined based on the general structure of a labeled graph (i.e. nodes, relations, properties, and values). Their instantiation depends on the event graph model in which the process data is stored. In the following, we present the solution we proposed for defining and instantiating intent (Section 4.6.1) and entity (Section 4.6.2) patterns.

4.6.1 Intent recognition

To perform a machine learning-based intent detection, the possible intent classes, that the system learns to detect, should be defined. For each of the NL categories presented in Section 4.5, a set of intent classes should be defined. The developed pipeline supports *content and behavioral queries*. Performance queries are left for future work. In the following, we define a set of general intent patterns that are instantiated according to the event property graph, for content and behavioral queries.

(i) Content Intent patterns

Content intent patterns are associated to content NL queries which allow the user to either

ask about particular nodes without taking into account any of their relationships, or about particular nodes that are involved in specific relationships with other nodes. For instance, in the question *'Which applications have a requested amount more than 15000'*, it searches for the Application nodes whose amount property is greater than 15000. In the question *'Which accepted applications have a requested amount more than 15000'*, it searches for the Application nodes connected, through the relation Affect, to the Activity nodes whose name is A_accepted.

Therefore, we define two classes of content intent patterns: Node patterns and Node_Relation patterns as shown in Table 4.1 (P1 and P3). Both patterns indicate that the user is seeking information about the node or its properties (as will be discussed in Section 4.7). Node intent pattern is associated with node matching queries whereas Node_Relation intent pattern is associated with relationship matching queries (i.e. Cypher query groups defined in Section 3.4.3). Given the event property graph in Figure 3.6, these patterns can be instantiated as shown in the intent instances column of Table 4.1. For example, the intent pattern Node is instantiated to each node type in the event property graph, e.g. intents Actor, Activity, Application and Offer. Similarly, the intent Node_Relation is instantiated to every possible node and relationship type [1], e.g. intents Actor_Contribute, Activity_Affect, etc. It is worth noting that for Node_Relation pattern instantiation, we add all possible pairs regardless of the relation direction. For instance, the intents Actor_Contribute and Activity_Contribute are two possible instantiations. The former is linked to queries about the actors executing a specific activity while the latter is linked to queries about the activities executed by specific actor. Examples of questions for each of these pattern instances are illustrated in Table 4.2.

	Intent pattern	Intent Instances	Cypher matching query
P1	Node	Actor, Activity, Application, Offer	node queries
P2	Agg_Node	Count_Actor, Count_Activity, Count_Application, Max_Application, etc.	node queries
P3	Node_Relation	Actor_Contribute, Activity_Contribute, Activity_Affect, Application_Affect, etc.	relationship queries
P4	Agg_Node_Relation	Count_Actor_Contribute, Count_Activity_Contribute, Max_Offer_Canceled, Avg_Offer_Offered, etc.	relationship queries

Table 4.1: Content intent patterns and their possible instantiations for the event property graph model in Figure 3.6

Two additional intent patterns, Agg_Node and Agg_Node_Relation (P2 and P4 in Table 4.1), are defined on top of the Node and Node_Relation patterns. These two patterns are similar to the first two patterns, but instead of reflecting the user's want for information about a node or its properties, they indicate that the user inquires for the result of one of the aggregate functions: maximum, minimum, average, count or sum. For instance, the question *'What is the maximum requested amount for car loan applications?'*, the user inquires about the result of the max function applied on the property amount of the Application nodes

[1] excluding the FollowedBy and {ObjectType}_FollowedBy relations as they have dedicated behavioral intents

Content Intent instance	Example questions
Activity **(P1)**	Which activities were executed today?
Application **(P1)**	Which applications have requested an amount of more than 15000?
Count_Activity **(P2)**	How many activities were executed last month?
Max_Application **(P2)**	What is the maximum requested amount for those applications with loan goal Home improvement?
Actor_Contribute **(P3)**	Who was involved in processing Application_2110141037? Who cancelled the applications with loan goal Home improvement?
Application_Affect **(P3)**	Which application with type New credit was incomplete? Give me the application ID of all canceled application.
Max_Offer_Canceled **(P4)**	What is the highest number of offers canceled in a single application?
Count_Activity_Contribute **(P4)**	What is the total number of UserAG contributions in Application_1765444083?
Actor_HasContributed_count **(P4)**	How many actors were involved in processing Application_2110141037?

Table 4.2: Examples of content questions and their associated intents

whose Goal is equal to 'car loan'. Its associated intent is Max_Application which is instantiated from the intent pattern Agg_Node. The intent does not specify the property on which the aggregate function should be applied. This information is inferred by the query construction component (more details in Section 4.7). On the other hand, the question *'Give me the highest number of offers canceled in a single application'* requires computing and returning the maximal number of Offer nodes connected to each Application node through the relation Canceled. Its associated intent is Max_Offer_Canceled which is instantiated from the intent pattern Max_Node_Relation.

(ii) Behavioral Intent Patterns

Behavioral intent patterns are associated to behavioral queries used to query the FollowedBy and {ObjectType}_FollowedBy relations in the event property graph. These queries provide answers to questions about the chronological order of the activities that were executed. They are central to many process analytics in general, and process mining in particular (e.g. all process discovery techniques require querying the behavioral aspect of processes [9]).

We classified the behavioral intents into two groups: intents under the relationship queries (P5 and P6) and intents under the path queries (P7, P8, P9 and P10) as shown in Table 4.3.

Node_Relation and Agg_Relation intent patterns indicate that the user is looking for the directly follow relation between two activities which can be either FollowedBy or {Object}_FollowedBy. They are instantiated into exactly the following three intent instances: Activity_FollowedBy, Object_FollowedBy and Count_FollowedBy. These intents belong to relationship matching queries which consist of selecting pairs of activities connected through FollowedBy or {Object}_FollowedBy relations. Activity_FollowedBy indicates that the user is seeking for the activity that directly precedes or follows a specific activity. Object_FollowedBy indicates that the user is seeking for the objects for which two specific activities follow each other. Finally, Count_FollowedBy indicates that the user is seeking for the number of times an activity precedes or follows another activity. Examples of NL queries assigned with these

	Intent pattern	Intent Instances	Cypher Matching Query
P5	Node_Relation	Activity_FollowedBy, Object_FollowedBy	relationship queries
P6	Agg_Relation	Count_FollowedBy	relationship queries
P7	Node_Relation*	Activity_FollowedBy*, Object_FollowedBy*	path queries
P8	Agg_Relation*	Count_FollowedBy*	path queries
P9	Path_Relation*	Path_FollowedBy*	path queries
P10	Length_Relation	Length_FollowedBy*	path queries

Table 4.3: Behavioral intent patterns and their possible instantiations for the event property graph in Figure 3.6

Behavioral Intent instance	Example questions
Activity_FollowedBy **(P5)**	What activity follows the submission of "App1" ?
Activity_FollowedBy **(P5)**	What activities follow the creation of offers?
Object_FollowedBy **(P5)**	Which applications include A_Cancelled directly followed by A_Submitted ?
Count_FollowedBy **(P6)**	How many times does the O_Cancelled activity directly follow the O_Created ?
Activity_FollowedBy* **(P7)**	What activity eventually follows A_Submitted in case "App1" ?
Object_FollowedBy* **(P7)**	Which offer entities contain an O_Cancelled that eventually follows the O_Created ?
Count_FollowedBy* **(P8)**	How many times does O_Cancelled activity eventually follow the O_Created ?
Path_FollowedBy* **(P9)**	Give me the trace of executed activities in "App1".
Length_FollowedBy* **(P10)**	What is the length of the trace in "App1"?
Length_FollowedBy* **(P10)**	How many activities were executed between O_Created and O_Cancelled in each offer?

Table 4.4: Examples of behavioral queries and their associated intents

intents are shown in Table 4.4 (P5 and P6).

Node_Relation* and Agg_Relation are similar to the Node_Relation and Agg_Relation patterns, but instead of looking only at the directly follow relation, they indicate that the user is looking for the eventually follow relations. They are instantiated similarly to the first two patterns. These intents belong to path matching queries which consist of selecting two activities connected to each other through FollowedBy or {Object}_FollowedBy relations with a variable length (denoted with the '*' symbol). Activity_FollowedBy* indicates that the user inquires about all activities that were executed eventually before or after a specific activity. Object_FollowedBy* indicates that the user inquires about the objects for which two specific activities eventually follow each other. Count_FollowedBy* indicates that the user inquires about the number of times an activity eventually precedes or follows another activity. Examples of NL queries assigned with these intents are shown in Table 4.4 (P7 and P8).

Path_Relation* and Length_Relation are intents dedicated to querying (sub-)traces which are fundamental to process mining techniques. A trace is simply the entire path of

NL query	Return all car loan applications validated within the last three months, and which received 3 offers minimum, order them by their requested amounts.
Intent	Application_Affect (**P3**)
NER	{(applications, (Node, Application)), (offers, (Node, Offer)), (requested amounts, (Node.property, Application.Amount)) (car, (Node.property.Value, Application.Goal.'Car')), . (applications validated, (Node.property.Value, Activity.Name.'A_validated')), (last three months, (date, [2021-12-01, 2022-03-01])), (3, (number,3)}
Cypher Query	**MATCH** (application: Application) - [:Affect] - (activity: Activity), (activity) - [:Affect] - (offer), (offer:Offer) - [:Offered] - (application) **WHERE** application.Goal= 'Car' AND activity.Name= 'A_Validated' AND activity.Time>='2021-12-01' and activity.Time<='2022-03-01' **WITH** application, COUNT(offer) as offerCT **WHERE** offerCT>=3 **RETURN** (application) **ORDER BY** application.Amount

Table 4.5: An example of a question with detected intent and extracted entities, as well as the corresponding Cypher query.

activities executed for a specific object. For example, in the event property graph in Figure 3.6, the entire path between A_created and A_cancelled is a trace for the Application object whose ID is app1. This is because all activities in the path are connected through Application_FollowedBy relation having the same value app1 for the relation property ObjectID. These intents are instantiated into Path_FollowedBy* and Length_FollowedBy*. The former indicates that the user is asking about the trace or sub-trace between two specific activities, while the latter indicates that the user is asking about the length or number of activities of a (sub-)trace. Examples of NL queries assigned with these two intents are shown in Table 4.4 (P9 and P10).

4.6.2 Entities extraction

Similarly to intent recognition, to perform a machine learning-based NER, we need to define the classes of entities that the NER system should learn to extract.

We define six possible patterns for the entities: **Node, Relationship, Node.property, Node.property.Value, Relationship.property and Relationship.property.Value**. We also add two additional patterns, **Date and Number**, which are associated to numerical and date terms that appear in the NL query. These two patterns refer to numerical and date values of node/relationship properties in the event property graph. They are processed by the query construction component to determine the property to which they refer.

In the following, we will use the term entity type to refer to an entity pattern and entity values to refer to their instantiations according to the event property graph. Node

and relationship entity types have the nodes' and relationships' labels as entity values, e.g. Activity, Actor, Application, Contribute, Affect. property entity types have the node/relationship property name as entity value, e.g. Activity.Name, Application.Amount, Application_FollowedBy.ObjectID. Finally, value entity types have the node/relationship property values as entity values, e.g. Activity.Name.'A_validated', Application.Goal.'Home improvement', Application_FollowedBy.ObjectID.'app1'. For instance, in Figure 4.2, the term validated with entity type Node.property.Value, its entity value indicates that the node is Activity, the property is Name, and the value is A_validated (written as Activity.Name.'A_validated'). The system also detects that 'applications' which correspond to the entity type Node, its corresponding entity value is Application.

The formal definitions of entity type, entity value and entity are given below.

Definition 4.1 (**Entity type**). *An entity type E_t can be one of the following: Node, Node.property, Node.property.Value, Relationship, Relationship.property, Relationship.property.Value, Number, Date. We denote by E_t^s the set of entity types.*

Definition 4.2 (**Entity value**). *Let $G_E = (N, R, \gamma, \lambda, \rho, \sigma)$ be an event property graph. For a node or relationship $e \in N \cup R$, $\lambda(e)$ is the node or relationship label; $att_e \in \rho(e)$ is the property name of the node or relationship, and $\sigma(e, att_e)$ is the property's value. An entity value E_v is defined as follows:*

- *$E_v \in Rng(\lambda)$ if $E_t \in \{$Node, Relationship$\}$; Rng returns the range of the function;*
- *$E_v \in \{\lambda(e).attr_e \mid e \in N \cup R \ \wedge \ attr_e \in \rho(e)\}$ if $E_t \in \{$Node.property, Relationship.property$\}$;*
- *$E_v \in \{\lambda(e).attr_e.\sigma(e, attr_e) \mid e \in N \cup R \ \wedge \ attr_e \in \rho(e)\}$ if $E_t \in \{$Node.property.Value, Relationship.property.Value$\}$;*
- *$E_v \in \mathbb{R} \cup \mathbb{D}$ if $E_t \in \{$Number, Date$\}$; where $\mathbb{R}$ is the set of real numbers and $\mathbb{D}$ is the set of date values.*

We denote by E_v^s the set of all possible entity values for G_E.

Definition 4.3 (**Entity**). *Let $G_E = (N, R, \gamma, \lambda, \rho, \sigma)$. An entity $E = (E_t, E_v)$ is a pair where E_t is the entity type as given in Definition 4.1, and E_v is the entity value as given in Definition 4.2. We denote by E^s the set of all possible entities for G_E.*

Definition 4.4 (**NER**). *Given an NL query Q_{NL} and a NER model trained with a set of NL queries and their associated entities E^s. The system output is $NER = \{(term, E) \mid term \in Q_{NL} \ \wedge \ E \in E^s\}$ which corresponds to the set of extracted terms from Q_{NL} associated with their entities.*

Table 4.5 shows an example of an NL query, its detected intent, and extracted entities.

4.7 Query construction component

The purpose of this component is to construct the corresponding Cypher query based on the detected intent and extracted entities from the NLU component. Algorithm 1 shows the pseudo-code of the query construction algorithm. It takes as input the NL query Q_{NL}, the detected intent I, and the extracted terms from the NL query associated with their entities NER (see Definition 4.4). It produces as output the corresponding Cypher query Q_C. The algorithm proceeds in four main steps. The first three steps allow to i) generate the MATCH-RETURN clause, ii) detect and generate the conditions that should be added to the WHERE clause and/or the WHERE part of the WITH clause and iii) detect whether WITH and ORDER-BY clauses need to be added and generate them. The resulting query may be syntactically incorrect. Therefore, the last step verifies and completes the query to generate a syntactically correct one. These different steps are detailed in the following sections.

Algorithm 1 Query construction pseudo-code

1: **Input:** $Q_{NL}, I, NER = \{(\text{term}, E)\}$
2: **Output:** Q_C
3: $Q_C \leftarrow$ " "
4: $construct\_match\_return(I, Q_C)$
5: $construct\_where(Q_{NL}, NER, I, Q_C)$
6: $construct\_with\_orderBy(Q_{NL}, NER, Q_C)$
7: $verify\_complete(Q_C)$

4.7.1 MATCH and RETURN clauses

MATCH-RETURN clauses are generated based solely on the detected intent I. Table 4.6 shows the templates of the MATCH and RETURN clauses associated with each intent pattern.

MATCH clause: The intent pattern from which I is instantiated allows to determine the Cypher query group (node, matching or path) (as detailed in Section 4.6.1) and consequently determine the pattern of the MATCH clause. It consists of either a single node or a sub-graph. In case I is an instance of the *Node* or *Agg_Node* patterns (i.e. node matching queries), the MATCH is made up of only one node type which is represented by I. For instance, the intent of the first question in Table 4.2 is *Activity*. Therefore, `MATCH (activity:Activity)` is constructed.

In case I is an instance of the *Node_Relation*, *Agg_Node_Relation* or *Agg_Relation* patterns (i.e. relationship matching queries), the MATCH is made up of two nodes connected through a relation. For instance, the intent of the question *'who was involved in process App1?'* is *Actor_Contribute*. Therefore, `MATCH (actor:Actor)-[:Contribute]-(activity:Activity)` is constructed, as the *Actor* node is connected through the *Contribute* relation to the *Activity* node. For the *Agg_Relation* pattern, the relation type only is represented by I, and the two nodes connected through this relation are inferred automatically. For instance, the intent of the question *'How many times does the O_Cancelled activity directly follow the O_Created'*

Cypher matching query	Intent pattern	MATCH-RETURN clauses templates
Node queries	Node	MATCH (n: Node) RETURN n or RETURN n.property
	Agg_Node	MATCH (n: Node) RETURN AGG(n) or RETURN AGG(n.property)
Relationship queries	Node_Relation	MATCH (n: Node)-[:Relation]-(m) RETURN n or RETURN n.property
	Agg_Node_Relation	MATCH (n: Node)-[:Relation]-(m) RETURN AGG(n) or RETURN AGG(n.property)
	Agg_Relation	MATCH (n)-[:Relation]-(m) RETURN COUNT(*)
Path queries	Node_Relation*	MATCH (n: Node)-[:Relation*]-(m) RETURN n or RETURN n.property
	Agg_Relation*	MATCH (n)-[:Relation*]-(m) RETURN COUNT(*)
	Path_Relation*	MATCH p= (n)-[:Relation*]-(m) RETURN longestPath(p)
	Length_Relation*	MATCH p= (n)-[:Relation*]-(m) RETURN LENGTH(longestPath(p))

Table 4.6: Intent pattern associated with their Cypher matching queries, and the general patterns of MATCH and RETURN clauses deduced from each intent pattern

is *Count_FollowedBy*. The intent specifies that the relation type is *FollowedBy* which connects two *Activity* nodes. Therefore, `MATCH(a1:Activity)-[:FollowedBy]-(a2:Activity)` is constructed.

Finally, in case I is an instance of *Node_Relation**, *Agg_Relation**, *Path_Relation** or *Length_Relation** patterns (i.e., path matching queries), the MATCH is made up of two nodes connected through variable length relations. This is denoted by the '*' symbol for the relation type. Similarly to the intent patterns of relationships pattern queries, one or both node types are inferred automatically or represented by I.

RETURN clause: Different types of information could be returned depending on the intent pattern: (i) If I is an instance of the intent patterns *Node*, *Node_Relation* or *Node_Relation** the node type with which the intent starts is returned (see example in Table 4.5). In some cases, when a user inquires about only some properties of a node, the system returns those properties rather than the entire node. Such a case can be recognized by examining the property names in E_T that do not have any corresponding value in the entity type. (ii) If I is an instance of the intent patterns *Agg_Node*, *Agg_Node_Relation* the result of one of the aggregation functions performed over the node in the intent or its property is returned.

(iii) if I is an instance of *Agg_Relation* or *Agg_Relation** the number of all selected paths is returned. (iv) if I is an instance of *Path_Relation** the longest selected path between two specific nodes is returned. (v) Finally, if I is an instance of *Length_Relation** the number of nodes in the longest selected path is returned.

4.7.2 WHERE clause

The goal of this step is to add the necessary conditions to the Cypher query by including the WHERE clause. The extracted entities in NER are added either as conditions in the query or as elements in the sub-graph to be matched. Each entity type in E^s that corresponds to node and relationship is added as sub-graph elements, while those corresponding to values are added as conditions.

WHERE clause: is made up of conjunction of triples[2] (property, operator, value) where property and value appear in the extracted entity types. Values can be of textual, numerical, or date types. Textual values in E^s are associated with an entity type *Node.property.Value* or *Relation.property.Value*. From their corresponding entity values, a condition is added by associating the value to the property using the "=" operator. In our example in Table 4.5, in the extracted entities (*car*, (*Node.property.Value*, *Application.Goal.'Car'*)): *Application.Goal.'Car'* is the extracted entity value where *Car* is the value of the property *Goal* of the node type *Application*. Therefore, the triple `application.Goal = 'car'` is added.

The same applies to activity name values. Except that in behavioral queries, the selected sub-graph consists of two activity nodes connected through one or more relations (e.g. `MATCH(a1:Activity)-[:FollowedBy]-(a2:Activity)`). The user may specify conditions on the value name of the first selected activity (i.e. a1) or/and on the second selected activity (i.e. a2). We will use the terms preceding and succeeding activities to refer to the first and last activities in the select sub-graph, respectively. For instance, in the question *'Which entities include A_Canceled executed directly after A_Validating? '*, *A_Canceled* and *A_Validating* represent the succeeding and preceding activities respectively.

To overcome this problem, a set of trigger words that indicate the presence of preceding (e.g., preceded, after) or succeeding (e.g., followed, before) activity is defined. The idea of trigger words/indicators was inspired by existing work (e.g. [60, 29]). We used online dictionaries to define the potential trigger words. Then we check whether one or more of these trigger words appear in the NL query. Once the trigger words were identified, we used the dependency parser in StanfordCoreNLP[3] library to determine the grammatical relationships between the trigger words and the activity names in the question. By inspecting these dependencies, we can determine whether the activity name corresponds to a preceding or succeeding activity. For instance, question Q5 in Table 4.7 contains a trigger word 'after'. The direct dependency between this word and A_Validating in the question indicates that A_Validating activity is a preceding activity. Following that, activity A_Canceled will be considered the

[2]The current version supports the conjunction of conditions
[3]https://stanfordnlp.github.io/CoreNLP/

succeeding activity.

Date values are automatically associated with the timestamp property of an activity node type. If a single date value is extracted, the "=" operator is used. Otherwise, if a date period is extracted, two conjunctions of triples are created with "≤" and "≥" operators.

To address the challenge **Chal_3** related to the numerical conditions, existing works in NLI systems make assumptions about the order of appearance of the triple (operator, property, value) in the NL query [29]. For example, they assume that the numerical value always comes directly after the operator's name. We do not impose such constraints on our work. Instead, we propose to find the valid (property, operator, value) by examining the grammatical relations between words.

To do so, we define trigger words to extract the operators from the question (e.g. 'at least', and 'minimal' in the question above are trigger words for the operator '≥'). The system tries every triple combination of property, operator, and numerical value that is possible. It then investigates the grammatical relationships between the elements of each triple. The types of relationships between words that should be considered were discovered empirically.

Table 4.7 shows examples of questions with grammatical dependencies between words, the possible indicators/ trigger words extracted, and the deduced conditions. The first row of the table contains the same example question stated above, with two extracted trigger words: minimum and greater than, which correspond to the ≥ and > operators, respectively. By examining the relationships between each operator, property, and number, we can conclude that (≥, monthly cost, 200) and (>, amount, 15000) are two valid triples, which are then translated into the conditions shown in the table.

4.7.3 WITH and ORDER-BY clauses

WITH clause: is constructed after the WHERE clause if there is a need for aggregation. We treat the presence of trigger words (e.g. in each, for each, by, etc.) that may indicate the need for an aggregation. The proposed algorithm examines the grammatical relationships, provided by the dependency parser, between the trigger word and each entity in the question. Then aggregates the results based on the entity which have a specific types of relationship with the trigger word. Question Q3 in Table 4.7 shows an example of an aggregation trigger word 'each' that is linked to the application entity. That is, the result should be aggregated according to the application node.

Furthermore, this clause defines the variable of the count function if there is a condition on it. For example in the example in Table 4.5, the WITH clause defines offerCT as the count of offers connected to each application. This variable is used after in the second WHERE clause to add the condition that it should be greater than 3. It is important to note, that the system recognizes that a WITH clause is required in this case, followed by a WHERE clause to add the condition, because the number 3 is associated with the offer entity rather than an property. That is, the number of offers should be counted (as defined in the WITH clause),

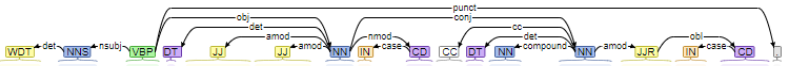
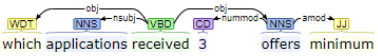
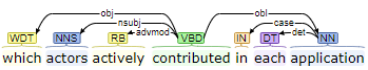
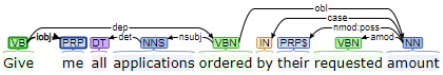
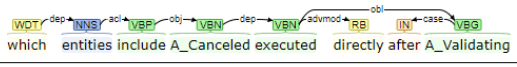
QuestionID	Dependencies	Trigger words/ Indicators	Conditions
Q1		minimum greater than	monthly cost>=200 amount>15000
Q2		minimum	count(Offer)>=3
Q3		each	aggregation by application
Q4		ordered	Order the result based on the amount
Q5		after	'A_Validating' predecessor activity

Table 4.7: Table of example questions with word dependencies, extracted indicators, and conditions inferred from these dependencies

and then the condition on the count is added (as defined in the second WHERE clause). The system handles conditions on count variables in the same way that it handles conditions on property values (e.g. Q2 in Table 4.7).

ORDER BY clause: is added if we detect trigger words indicating that the user requires the result ordered (e.g., ordered, descending, sorted, etc.). Similarly to the aggregation and numerical value and activity name conditions, the grammatical dependency with the trigger word determines the entity that we need to order the result based on it. For example in Table 4.7 question Q4, the trigger word is *ordered* and the requested amount is the entity that we should order the result based on it.

4.7.4 Cypher query verification and completion

The last step aims to address the challenge **Chal_4** by verifying and completing the Cypher query. First, ensure that the Cypher query is syntactically correct. Technically speaking, each node or relation whose property is conditioned in the *WHERE* clause or mentioned in the *WITH* or *ORDER BY* clauses should appear in the *MATCH* clause. Second, ensure that the sub-graph in the *MATCH* clause is complete. In other words, each pair of nodes that appear in the *MATCH* clause, and that are connected through relation in the event graph meta-model depicted in Figure. 3.5, should be also connected through the same relation in the *MATCH* clause. For instance, for the question *'Who was involved in processing App1?'* the intent is *Actor_Contribute*. Therefore, as explained before the *MATCH* clause: `MATCH (actor:Actor)-[:Contribute]-(a:Activity)` is constructed. The *Application* node should be also added to the *MATCH* clause, as it appears in the extracted entities (i.e. *App1*). As a result, the resulting *MATCH* will be: `MATCH (actor:Actor)-[:Contribute]-(a:Activity),`

`(app:Application)`. However, without connecting the *Activity* to the *Application* node, the Cypher query will return the actors who contributed to all activities, without taking the application 'App1' into account. This does not provide an answer to the user's question. Therefore, a relation *Affect* should be added between *Activity* and *Application* nodes to connect the subgraph. This way, all implicit nodes and relations required to complete the constructed Cypher query are added.

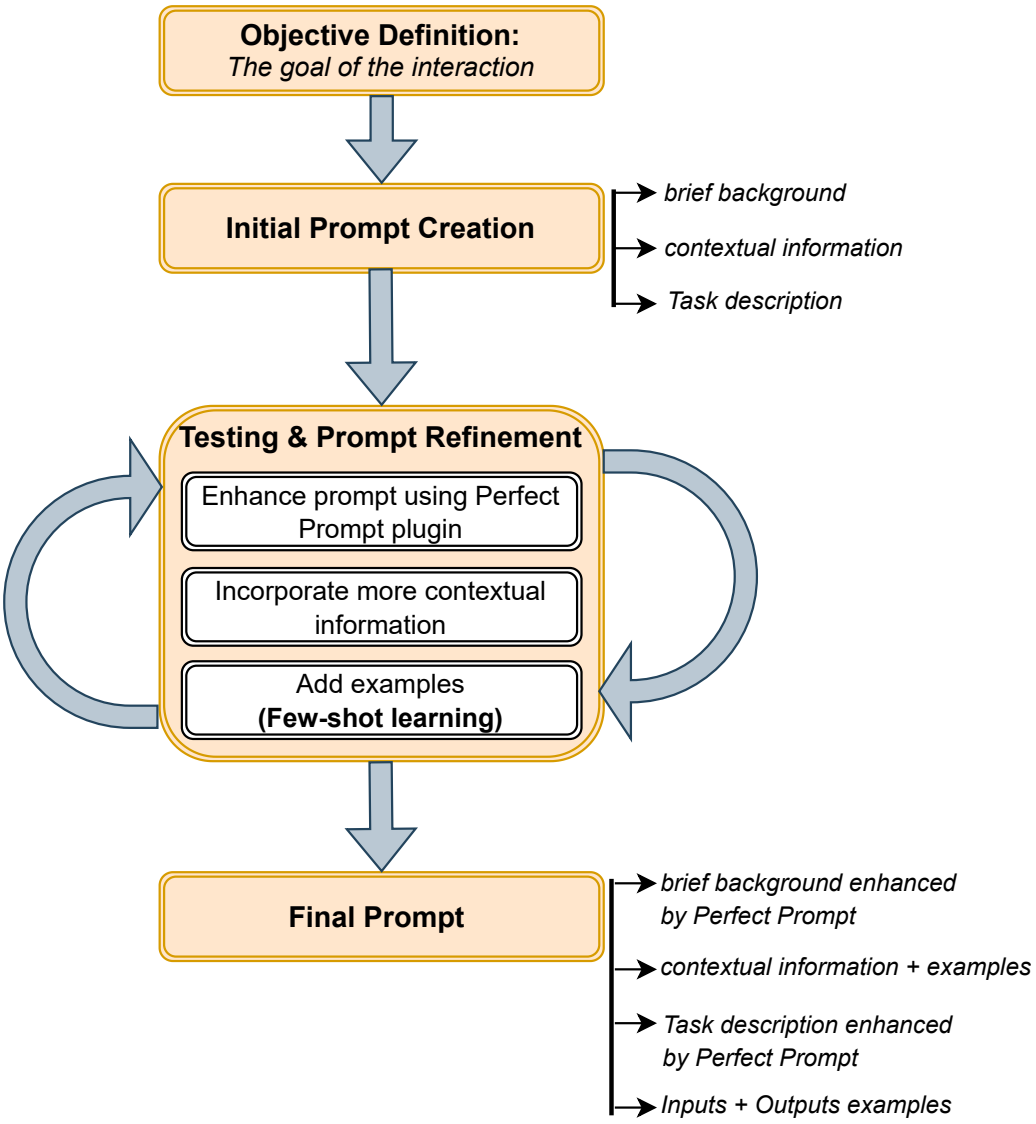


Figure 4.3: Prompt Engineering process for optimizing LLM responses

4.8 Automated generation of NL training data

Intents and entities are clearly domain-specific and instantiated depending on the process data. Accordingly, the generation of training data is required when switching from one process domain to another. This manual effort restricts the pipeline's applicability to specific domains. To overcome this limitation, we propose an automated approach to generate NL queries using prompt engineering and LLMs. Prompt engineering involves crafting carefully designed instructions or prompts to guide the LLM's output toward the desired behavior. By utilizing LLMs like GPT-4, we can leverage its powerful language generation capabilities to automatically generate a wide range of NL queries without the need for a manually annotated dataset for each new domain.

Figure 4.3 illustrates the prompt engineering process that we designed to create well-crafted prompts for enhancing the LLM's response quality. The process begins with the "Objective Definition" step, where the specific goal of the interaction with the LLM is determined. This involves precisely specifying the type of responses sought from the model. After defining the objective, the second step is the "Initial Prompt Creation." In this step, an initial prompt is crafted based on the defined objective. The prompt should ensure clarity, specificity, and relevance to guide the model toward generating the desired response. It may include a brief and general background to provide context, as well as essential contextual information to help the model understand the topic at hand. Additionally, a task description is included to guide the model in generating the expected response.

The third step involves an iterative process of "Testing & Prompt Refinement." In this phase, the initial prompt is put to the test, and its effectiveness in guiding the model's response is evaluated. The Perfect Prompt plugin is utilized to optimize the prompt and improve the quality of GPT-4's responses. The plugin evaluates the user's input and, if needed, rephrases it to make it clearer, more specific, and contextually appropriate. During the iterative refinement process, relevant contextual information may be incorporated into the prompt to narrow down the scope of the response and align it better with the user's intent. Additionally, providing examples of user inputs and their expected outputs can aid in fine-tuning the model's responses and making them more accurate and relevant. Once the iterative refinement process is completed, a high-quality prompt is obtained, and it is used in interactions with the LLM.

To achieve automated NL generation, we follow the prompt engineering process described above. Our goal is to create a set of general NL templates for each intent. These templates serve as blueprints for generating NL queries and include placeholders that refer to specific properties, property values, and other domain elements. By populating these placeholders with actual values from the event property graph of the domain, we can dynamically create NL queries tailored to various intents.

Once we defined our objective, we proceeded to create the initial prompt. This prompt includes a comprehensive description of our pipeline, with a particular emphasis on the NLU component. We highlighted general intent patterns and demonstrated their adaptation to

specific graph models. In addition, we delve into the task of template generation from intents and generating NL templates for each intent. The prompt underwent continuous refinement and testing, evaluating responses obtained from GPT-4. We provided GPT-4 with valuable information related to each intent, including concise textual descriptions, properties relevant to the intent, possible property values, and more. Additionally, we supplied GPT-4 with exemplary templates for various intents, along with examples of NL queries generated from these templates. Through an iterative process, we achieved a high-quality prompt, utilized to interact with the LLM and obtain the generated NL queries. An illustrative example of this prompt is provided in Appendix A.

Figure. 4.4 illustrates the interaction step with the GPT-4 model. In this step, we ask GPT-4 to generate, for each intent, more than 50 NL queries from its associated templates. This is achieved by replacing the placeholders with real values and employing paraphrasing techniques to diversify the generated queries. As a result, we obtain a substantial variation of NL queries, expressed in different syntax and structures, for each intent. These automatically generated queries constitute a vast and versatile input training dataset.

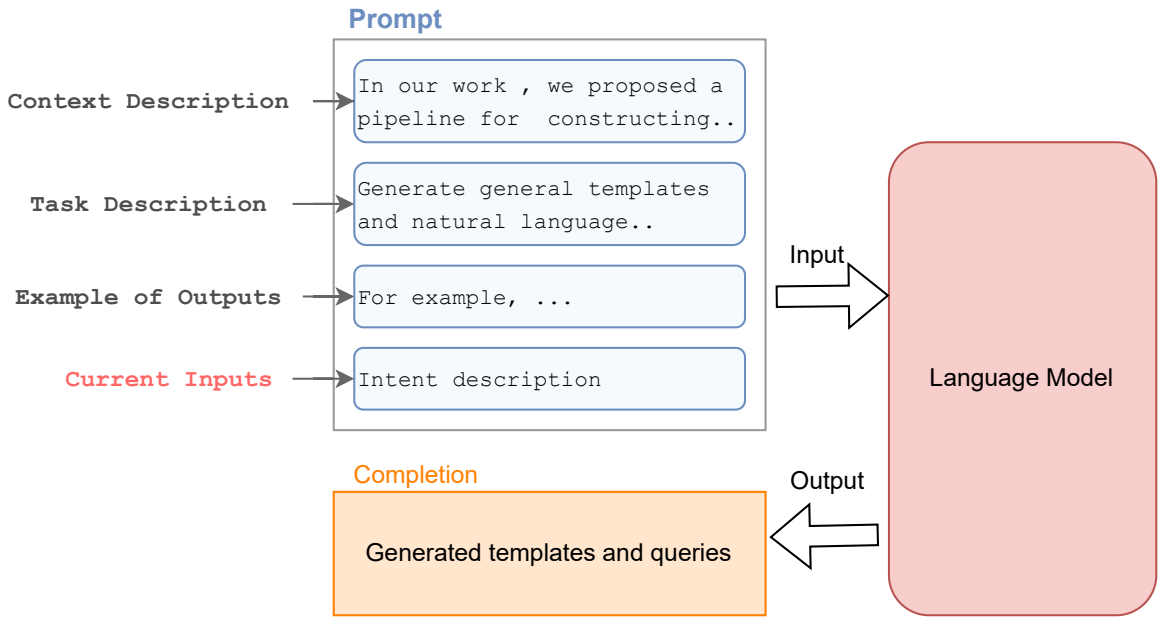


Figure 4.4: Prompt Engineering for NL queries generation

Considering the example of the *Application* intent instantiated from the graph model depicted in Figure 3.6. This intent allows users to inquire about applications with specific conditions. It has the following properties: application ID, requested amount, a loan goal. The property values for each property are as follows:

- **Requested amount:** A numerical value.
- **Application ID:** Has the form `Application_number`.
- **Loan goal:** Can have values from the set ['car', 'home improvement', 'investigation'].

Generated Intent Template:

The template for the *Application* intent consists of various NL query patterns with placeholders representing specific properties and values. Here are some of the generated intent templates:

1. "Give me all applications with their {application_properties}."
2. "Which applications have a {application_property} greater than {specific_value}?"
3. "List all applications with a loan goal of '{specific_loan_goal}'."
4. "What are the {property_1} and {property_2} of applications with a {application_property} less than {specific_value}?"
5. "How many applications are there with a {application_property} greater than {specific_value} and a loan goal of '{specific_loan_goal}'?"
6. "Which applications have an application ID starting with '{specific_prefix}'?"
7. "What is the {property_1} of application '{specific_application_id}'?"
8. "Which applications have a loan goal not equal to '{specific_loan_goal}'?"
9. "Give me the {property_1} of applications with a {application_property} less than {specific_value} and a loan goal of '{specific_loan_goal}'."
10. "What are the {property_1} of applications with a {application_property} greater than {specific_value} and a loan goal of 'car'?"

Generated NL Queries from Template:

Using the above intent templates, GPT-4 generates various NL queries for the *Application* intent. Here are some examples:

1. "Give me all applications with their requested amount, application ID, and loan goal."
2. "Which applications have a requested amount greater than 1000?"
3. "List all applications with a loan goal of 'home improvement'."
4. "What are the application ID and loan goal of applications with a requested amount less than 500?"
5. "How many applications are there with a requested amount greater than 2000 and a loan goal of 'car'?"
6. "Which applications have an application ID starting with 'App_2022'?"
7. "What is the requested amount of application 'Application_12345'?"

8. "Which applications have a loan goal not equal to 'investigation'?"

9. "Give me the application ID of applications with a requested amount less than 1000 and a loan goal of 'home improvement'."

10. "What are the requested amount of applications with a requested amount greater than 1500 and a loan goal of 'car'?"

By adopting this automated approach, we not only eliminate the labor-intensive manual data generation but also enhance the flexibility and adaptability of our pipeline. With the ability to handle a wide array of user queries across various domains, our NLU-powered pipeline becomes more robust and capable of delivering accurate and contextually appropriate responses to users.

4.9 Evaluation & Discussion

The approach has been implemented as a standalone Python application with a conversational interface. The application is connected to Neo4j which stores the event property graph. We evaluated our approach using two publicly available event logs that were already used in Section 3.5. As was previously mentioned the datasets are represented in CSV and OCEL format, and are related to two different process domains. For each process domain, we collected NL queries with lexical and syntax variations (Section 4.9.1).

We conducted separate experiments on the two components of the pipeline to evaluate various aspects in a controlled environment (Section 4.9.2 and Section 4.9.3). In Section 4.9.4, we discuss the threats to the validity of the performed experiments. The details related to the source code, data used, and obtained results are available at at `https://www-inf.telecom-sudparis.eu/SIMBAD/tools/ProcessNLI/`.

4.9.1 NL queries collection

In this evaluation, we used data from BPIC'17 and order management event logs, which contain data related to the loan application process and order management process. In order to collect NL queries related to the loan application process, a workshop was held with two different groups of Master students. The students were not familiar with the implementation, and are unaware of the Cypher language or how to access the stored data using graph queries. First, a brief overview of the loan application process was provided to the students in order for them to understand the general steps involved (i.e. activities of the loan application process). Then, various aspects of analysis were explained, with numerous example questions. The NLI is then provided to the students, who were asked to formulate questions using different syntaxes and vocabularies to analyze the process execution data.

We filtered the NL queries at the end of the workshop to remove the ones that are not yet supported by our system (i.e. complex queries that require for example sub-queries and negation, queries related to the performance category, etc.). As a result, we ended up with more than 300 content queries related to 16 different intents, and more than 70 behavioral queries related to 8 different intents (details about the NL queries and the intents are available in the provided link).

In addition, we used a paraphrasing tool[4] to collect NL queries about the order management process. The paraphrasing task provides syntax and lexicon variations of a NL text without changing the original meaning. We used the tool to paraphrase a set of manually generated questions. In the end, 65 content queries related to 9 different intents, and 85 behavioral queries related to 8 different intents were generated (for details see the provided link).

4.9.2 Experiments on NLU component

We conducted two major experiments on the NLU component to justify the use of ML for intent detection and entity extraction. The main disadvantage of using machine learning, as discussed in the related works section, is that it requires a training dataset, which is not always available. In this experiment, we aim to evaluate the performance of a machine-learning model for detecting intent and extracting entities using a small training dataset. Second, we compared the results of the machine learning model for intent detection to those of a rule-based approach.

4.9.2.1 Experimental setup

We used Wit.ai[5], a service, for performing intent detection and entity extraction. For each process domain, a machine learning model in Wit.ai is trained with typically a small set of utterances labeled with their corresponding intents and their associated entities as explained in Section 4.6.1 and 4.6.2. The training utterances were created by hand. For the BPIC'2017 event log, we trained the model with 390 utterances about the loan application process. The model is then tested with more than 380 NL queries collected from external users. As for the order management log, we trained the model with 240 utterances about the order management process, and we test it with 150 NL queries generated using a paraphrasing tool (as detailed in Section 4.9.1).

In the second experiment, we compared the detected intents using the machine learning model in Wit.ai to a rule-based approach. The rule-based approach was implemented separately and attempts to detect the corresponding intent based solely on the extracted entities and defined trigger words. The rules were devised with high precision. They cover a wide range of cases, particularly those deduced from the NL queries used in the machine learning

[4]Paraphrasing tool: https://quillbot.com/
[5]https://wit.ai/

model's training data. Trigger words are defined and used to determine whether the user is asking about a specific type of information. For instance, the trigger words *[path, trace, sub-trace, etc.]* indicate that the user is asking about a trace. Thus, the detected intent will be *Path_FollowedBy**. In addition, rules are defined to take into account the properties or elements that appear in the NL queries and use these elements to predict the corresponding intent. For instance, in the question *'What are the amounts of all applications?'*, the extracted entities *amounts* and *applications* are exclusively associated with the Application node, hence the recognized intent is Application.

As evaluation metrics, we used accuracy for intent detection and precision/ recall/ F-score for entity extraction. The accuracy is computed as the number of questions with correctly detected intent divided by the total number of NL queries. For each NL query, the precision/ recall/ F-score of entities extraction is computed. The precision for a given NL query is calculated by dividing the number of correctly extracted entities by the total number of entities extracted. The recall is calculated by dividing the number of correctly extracted entities by the total number of entities expected to be extracted. The metrics are then averaged over all NL queries.

4.9.2.2 Results

		Content queries	Behavioral queries	Average
BPIC'2017 log	Intent	acc= 0.731	acc= 0.756	acc= 0.743
	Entities	prec= 0.901 rec= 0.908 F-score= 0.902	prec= 0.888 rec= 0.917 F-score= 0.899	prec= 0.894 rec= 0.912 F-scrore= 0.90
Order Management log	Intent	acc= 0.677	acc= 0.624	acc= 0.65
	Entities	prec= 0.952 rec= 0.947 F-score= 0.947	prec= 0.988 rec= 0.955 F-score= 0.967	prec= 0.97 rec= 0.951 F-score= 0.957

Table 4.8: Wit.ai evaluation results for intent recognition and entity extraction in both datasets for content and behavioral queries

The results of the intent recognition and entity extraction obtained using the machine learning model in Wit.ai for content and behavioral queries in each dataset are shown in Table 4.8. The results indicate that the machine learning model produced significant results.

The NL queries related to the BPIC'2017 event log yield, in average, an accuracy of 0.743 for intent detection, while the NL queries related to the order management log yield in average an accuracy of 0.65. This distinction is due primarily to the type of the NL queries employed in the evaluation. Indeed, the NL queries related to the BPIC'2017 event log were obtained from external users, who may have, in some cases, similar ways of asking a question, reducing NL variations. The NL queries related to the order management log, on the other hand, were created using a paraphrase tool. The paraphrasing tool has the ability to alter the syntax and use lexicons that are rarely used by users. For instance, to inquire about the number of

a given entity, users commonly use terms such as *[what is the number, how many, etc.]*. The paraphrasing tool, on the other hand, generates other expressions like *how frequently, etc.*, which may confuse the model and subsequently produce incorrect results.

At the level of entity extraction, a high F-score is obtained for both datasets. By closely inspecting the results, we were able to underline the following observations. First, we analyzed the low precision values which indicate that some entities were incorrectly tagged. This was mainly due to that some entities may have common keywords/synonyms. For example, given the question *'What is the loan goal of application with amount 7500?'*, and the extracted term *amount*, there are two entities with which *amount* can be tagged: 1) the property *Requested Amount* of the artifact node *Application* or 2) the property *Offered Amount* of the artifact node *Offer*. Similarly, in the order management dataset, given the question *'What is the most expensive product?'*, the *expensive* term could refer to: 1) the maximum *cost* of the artifact node *Product* or 2) the maximum *price* of an *Activity* node. By looking at the question context, it becomes trivial to which option the term should be associated with. However, the machine learning model was unable to consider the NL query context and assign the right entity name accordingly.

Second, we discovered that the incorrectly extracted activity names are responsible for more than 50% of the incorrectly extracted entities in the BPIC'2017 dataset. This was primarily owing to the dataset's similar activity names. For instance, the verb 'canceled' could refer to two different activity names: *A_Cancelled* and *O_Cancelled*. Therefore, only using the verbs is not sufficient to properly extract the activity name. Instead, a set of expressions should be utilized to train the model to determine the appropriate activity. For instance, the expressions: canceled application, cancellation of the application, cancel the application, etc. all refer to the activity name *A_Cancelled*. As a result, detecting activity names involves more than just a few synonyms; rather, it involves a large number of expressions that could refer to the same activity name, making it difficult to train the model with all of them. The order management log, on the other hand, does not have this difficulty because the activity names are indistinguishable and each is referenced with a specific verb.

	BPIC'2017 log		Order Management log	
	Content queries	Behavioral queries	Content queries	Behavioral queries
Intent detection with ML model	**0.731**	**0.756**	**0.677**	**0.624**
Intent detection with rule-based	0.662	0.589	0.569	0.47

Table 4.9: Accuracy for intent detection using machine learning model in Wit.ai vs rule-based approach for both datasets

For the second experiment, Table 4.9 shows the results of the intent detection accuracy using Wit.ai's machine learning model versus using the rule-based approach. In both datasets, the machine learning model in Wit.ai clearly outperforms the rule-based approach for detecting intents for content and behavioral queries. This was primarily due to two factors. First, the dependence of the rule-based approach on the extracted entities. Incorrectly extracted entities result in incorrectly assigned intents. For example, the rule-based approach detects

that the question *'What are the amounts offered in App1?'* begins with the term *amounts*. However, as previously stated, the term *amounts* may be associated to two different entities: *Amount* property within *Offer* node, or *Amount* property within *Application* node. Once the entities are incorrectly extracted, the detected intent will be definitively incorrect.

Second, the trigger words used by the rule-based approach could be in some cases perplexing. For example, if a user inquires about the maximum of a specific property, the system employs trigger words such as maximum, maximal, and so on. This is correct if the question is *'What is the maximal requested amount?'*. On the other hand, these same words could be used to specify other purposes. For instance, in the question *'What are the applications with a maximum amount of 15000'*, the word 'maximum' refers to the less than operator. It denotes that the amount property should be less than 15000. Therefore, the system will be confused, and an incorrect intent will be assigned.

As a result of the above experiments, we made the following conclusions. First, the machine learning model correctly recognizes process intents and entities without the need for a large training dataset. Second, developing rules to support NL diversity takes time and effort. In addition, they are in some cases, unable to produce accurate results due to their reliance on the extracted entities and human-defined trigger words. Machine learning, on the other hand, is more robust to NL variation. It outperforms the rule-based approach in two different process domains, despite the small size of the training data.

4.9.3 Experiment on query construction component

In this experiment, we aim to determine whether the query construction component is able to construct the right Cypher query from the detected intent and extracted entities. By design, our constructed queries are syntactically correct (details in Section 4.7.4). Therefore, we evaluate whether they are semantically correct (i.e. they return the correct result as inquired by the user). We compare the intent-based approach (i.e. which takes the detected intent and extracted entities to construct the Cypher query) to a baseline that does not involve an intent detection step (i.e. it does not take into account the detected intent). It is worth noting that, to the best of our knowledge, no existing works for automatically constructing a Cypher query that could be used for a comparative evaluation. That is why, we compared our intent-based system to a baseline by removing one critical step in the pipeline. The system in the baseline returns the information in the selected sub-graph that does not have any values. For instance, for the question *'What applications include O_Created activity directly followed by O_Canceled?'*, the constructed sub-graph includes conditions on the two activity names and miss conditions on the ObjectID property of the FollowedBy relation. As a result, the latter will be returned. In case a sub-graph has conditions on all of its elements, the entire selected sub-graph is returned.

4.9.3.1 Experimental setup

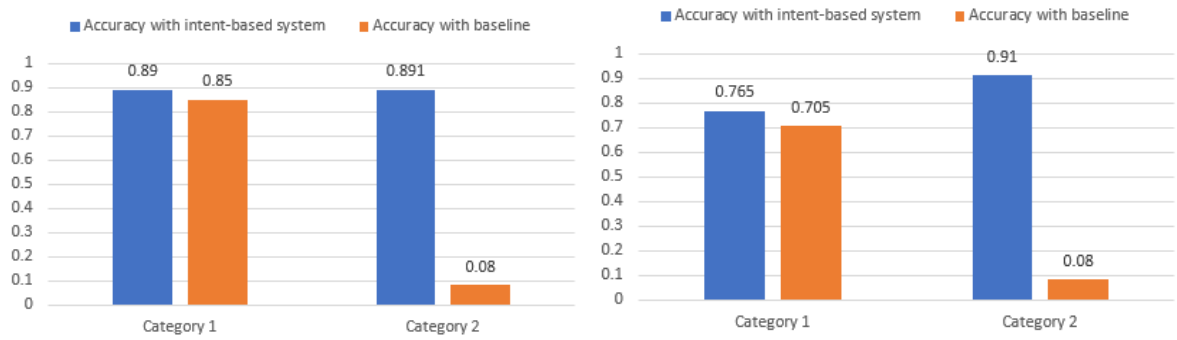
We selected the NL queries for which the detected intent and the extracted entities of the NLU component are correct. As a result, we obtained 202 and 82 NL queries related to the BPIC'2017 and the order management event logs respectively. The NL queries are grouped into two categories. The first category (i.e. *category 1*) consists of the NL queries that include the information to be returned in the extracted entities. The second category (i.e. *category 2*) consists of the NL queries that inquire about a specific type of information that is not present in the extracted entities. For example, these queries could inquire about an aggregation function (e.g., count, max, min), a direct relationship between activities, etc. These types of information are derived from the context of the question and are indicated by the intent in our intent-based system.

For each NL query, we examined whether the generated Cypher query returned the expected answer. As an evaluation metric, we computed the accuracy by dividing the number of semantically correct Cypher queries by the total number of queries.

4.9.3.2 Results

Figure 4.5 shows the accuracy of the query construction component in the intent-based system versus the baseline using the BPIC'2017 log (Figure 4.5a) and the order management log (Figure 4.5b). On average, the intent-based system has an accuracy of 0.89 and 0.84 using BPIC'2017 and order management logs respectively (blue columns in Figure 4.5). These results show that for two separate datasets, our system was able to generate the expected Cypher query with high accuracy. However, by analyzing the incorrectly constructed Cypher queries for the two datasets, we discovered two major reasons. The first is wrongly identifying and extracting numerical conditions from NL queries. The second is incorrectly assigning preceding or succeeding activities, particularly for behavioral queries. As stated in Section 4.7, numerical conditions are built by examining the grammatical relations between the entity, operator and value. Similarly, preceding and succeeding activities are found using trigger word definitions and evaluation of the grammatical relations between these trigger words and the activity names in the NL query. However, due to the wide variety of NL, it was difficult to cover all possible ways of asking a question, and subsequently, the possible grammatical relations that exist between these elements.

The query construction component in the intent-based system and the baseline achieve on average, a similar accuracy in the first category (i.e. Category 1) of NL queries in both datasets. This is due to the fact that these queries inquire about information included in the extracted entities. The intent-based system clearly outperforms the baseline in the second category (i.e. Category 2) of questions with an average accuracy of 0.849 versus 0.015 using the BPIC'2017 log, and an average accuracy of 0.91 versus 0.08 using the order management log. In this category, the intent assists the system in determining what information the user is seeking, and which can be deduced from the question context. Therefore, in some cases, entities are not enough to express the intention of the users.



(a) BPIC'2017 log **(b)** Order management log

Figure 4.5: The accuracy of the query construction in the intent-based system and the baseline

This experiment shows the importance of our system's intent detection step in determining the type of information the user is inquiring about from the entire query context. Additionally, it demonstrates that the defined intent and entity patterns and their instantiations, assist the system in constructing the right Cypher queries.

4.9.4 Threats to Validity

A potential threat to the validity of our study arises from the absence of a comparative analysis with existing works. This lack of comparison can be attributed to several factors:

- NLIs for Process Querying: When considering related works in the domain of NLIs for process querying, we find that the most relevant sources are those presented in [14, 70]. However, these existing approaches do not provide a clear categorization of the types of questions they support. Furthermore, they do not offer comprehensive results for each evaluated NL query, which makes it challenging to conduct a fair and meaningful comparison.

- NLIs to Databases: In the realm of NLIs to databases, other existing approaches focus on constructing SQL, SPARQL, or Cypher queries. However, these approaches typically utilize different datasets and benchmarks for their evaluations. Our data is process-oriented, which inherently differs in nature from the data used in these other techniques. Consequently, the techniques and evaluation objectives are distinct, making a direct comparison difficult.

Another threat to validity that could potentially impact the validity of our approach is its specificity to the proposed NLI and the associated graph metamodel used for storing process data. It is important to acknowledge that there can be various metamodels based on LPG for representing process-related information, as the one proposed in [55]. Therefore, it becomes imperative to generalize our pipeline in such a way that it can construct Cypher queries for not just one specific metamodel but for a range of LPG metamodels, with minimal manual

intervention required when transitioning from one metamodel to another. This flexibility and adaptability in query construction would ensure that our NLI can effectively handle diverse data representations and metamodels, making it more versatile and accessible to a broader range of users and scenarios. It would also enhance the scalability and applicability of our approach as the field evolves and new metamodels emerge.

4.10 Conclusion

In the prior chapter, we delved into the subquestion **(RQ1-3)** concerning the storage of process data. Now, in this chapter, we have successfully addressed the primary research question, **(RQ1)** for providing an automated solution to query process data using NL. This was achieved by examining the following supplementary questions:

- **RQ1-1**: How to provide an AI-based solution that automates the process of querying process data using NL?
- **RQ1-2**: What design principles can be implemented to create an approach that possesses sufficient generality, allowing for seamless transitions between diverse process domains with minimal manual intervention?

To answer these questions, we introduced an intent-based NLI tailored for querying process execution data. This interface simplifies the user experience by deciphering a user's intent from their NL queries. It then automatically formulates the corresponding Cypher query, which is executed against the process data stored in a graph database. The result is then returned to the user. Our innovative system integrates both machine learning and rule-based methodologies, resulting in a hybrid NLI model. This model comprises two primary components. The initial component, grounded in machine learning, NLU, which encompasses intent recognition and extraction of named entities. Following this, the second component, grounded in a rule-based approach, leverages the information garnered from the NLU component to construct the corresponding Cypher query.

Additionally, in order to address **(RQ1-2)** goal of creating a generalized solution, we introduced several strategies. Firstly, we outlined general patterns for intents and entities rooted in the LPG metamodel. These templates are subsequently instantiated in line with the event property graph model. As a result, the system can effortlessly determine potential intents and entities associated with NL queries related to each event property graph. Secondly, considering our machine learning component necessitates training datasets linking NL queries to specific intents and entities, we unveiled an automated strategy. This leverages prompt engineering coupled with LLMs to produce NL queries. Utilizing LLMs such as GPT-4, we can automatically generate a diverse range of NL queries, eliminating the need for manually annotated datasets for each domain transition.

To validate our proposed system, we applied it to two publicly available event logs from BPIC'17, as well as an order management log. We sourced NL queries from external users and

further expanded our testing pool using a paraphrasing tool. Our experiments yielded several notable findings: (i) the machine learning model is able to recognize intent and extract entities from typically a small training dataset, (ii) it outperforms the rule-based model for intent recognition and (iii) the intent increases the accuracy of the query construction component.

However, we acknowledge certain limitations within our approach:

- Evaluation aspect: The lack of a side-by-side comparison with pre-existing methods can be seen as a drawback. This gap is largely attributed to the disparity in question types, dataset variations, and evaluation standards across different methodologies.
- Graph model dimension: We concede that our proposed NLI is tailored to our specific graph metamodel designed for storing process data. However, there's a plethora of potential metamodels rooted in LPG that can depict process-related data. Hence, it's crucial to restructure our pipeline to ensure it can generate Cypher queries for a multitude of LPG metamodels. Ideally, this would necessitate minimal manual adjustments when shifting between different metamodels.

In addition, we acknowledge the importance of conducting a comparative evaluation to assess the performance of the NLU component when trained using automatically generated NL queries. Thus, we plan to perform a comparative evaluation. We aim to compare the detected intent and extracted entities obtained when the model is trained with manually constructed queries to the results achieved when the model is trained using the NL queries generated automatically.

Transitioning from an individual instance analysis to a more comprehensive process-level exploration, the subsequent chapter unveils our approach to tackle the research problem **(RQ2)**. Our objective is to simplify the discoverability and the accessibility of process mining methods through a service-oriented framework. This framework design endeavors to describe, design, and match process mining services with user needs expressed in NL.

Chapter 5

Service-Oriented Architecture for Discovering and Accessing Process Mining Techniques

Contents

5.1 Introduction

Process mining represents a rapidly expanding domain of research situated at the crossroads of data science and BPM [156]. By employing process mining techniques, organizations can

gain valuable insights into the behavior of their processes. These insights encompass the identification of inefficiencies, the detection of compliance breaches, and the pinpointing of opportunities for process optimization. The various categories of process mining techniques encompass discovery, conformance, enhancement and prediction. Despite the vital role that process mining methods play, data scientists and process analysts frequently encounter challenges when attempting to apply these techniques in practical scenarios. One of the foremost hurdles is the initial selection of an appropriate method tailored to meet their specific needs and requirements.

Another challenge is associated with the accessibility and applicability of these techniques. Typically, process mining methods are available either in the form of source code or as part of standalone software tools. However, accessing a method through its source code can be daunting for analysts without expertise in the relevant programming language or technical acumen to decipher intricate technical details. Additionally, certain methods are embedded within software tools that lack compatibility with other applications, significantly constraining their adaptability across diverse workflows and integration into custom applications.

As previously discussed in Chapter 2, prior efforts have aimed to tackle either the first or the second challenge. On one hand, to address the challenge of discovering process mining methods, surveys and systematic reviews (e.g., [106, 47, 9, 51]) have been conducted. These comprehensive studies compile existing methods within each field, analyze them, and provide comparisons based on identified concepts. These resources assist analysts in gaining a broad understanding of available methods and selecting the most suitable one for their specific needs. Nevertheless, even with these efforts, analysts still face difficulties in terms of accessing and applying the identified methods.

On the other hand, approaches have been proposed to tackle the issue of accessibility and integration of process mining techniques. These approaches include API-based solutions (e.g., PM4Py [27]) and web service-based solutions (e.g., PM4Py-WS [28], Everflow[1]). These solutions strive to enhance the accessibility and integration of process mining techniques. However, existing service-based solutions in the field of process mining have limitations, particularly in the realm of prediction. Another significant drawback is the absence of comprehensive descriptions for available services, making it challenging for analysts to easily discern service properties, required inputs, and expected outputs. Furthermore, to the best of our knowledge, there is no existing solution that aids analysts in the automatic discovery and accessibility of process mining-related services.

To address these challenges, this chapter introduces a service-based solution. The primary objective is to alleviate the difficulties faced by analysts when discovering and accessing process mining methods, with a primary focus on discovery, conformance, and prediction methods. The proposed solution harnesses REST APIs to deliver process mining services and consists of three key components:

- **Services Description:** This component employs a graph metamodel based on LPG [7]

[1] https://www.everflow.ai/

to comprehensively describe available discovery, conformance, and prediction methods. It encompasses essential concepts for representing services, such as service properties (e.g., prediction type, methodology, algorithm), as well as input and output requirements.

- **Unified Service-Oriented REST API Design:** Leveraging the inherent service-oriented nature of process mining methods, this component offers users a cohesive framework for accessing and utilizing various prediction functionalities, ensuring a unified experience.
- **Services Matching:** This component matches user requirements with suitable process mining services. It takes NL descriptions from users, constructs Cypher queries for execution over the LPG graph, and generates REST API calls for the selected methods.

The solution's validity has been established through a proof of concept, and a user experience evaluation was conducted by involving external users who utilized traditional methods, such as process mining tools and literature, to search for process mining methods. Furthermore, feedback from users regarding the REST API design was collected and evaluated. This research endeavor was formally presented in a submission to the IEEE Transactions on Service Computing. [2]

The remainder of this chapter is organized as follows. Section 5.2 introduces key concepts relevant to this chapter. Section 5.3 offers an overview of the proposed service-oriented architecture. The three components of the architecture are elaborated upon in Section 5.4, Section 5.5, and Section 5.6. Section 5.7 presents the proof of concept, while the evaluation of user experiences and the REST API design is expounded upon in Section 5.8. Finally, Section 5.9 concludes this chapter.

5.2 Basic concepts

In this section, we lay the foundation by introducing fundamental concepts pertinent to this chapter. Firstly, we provide an overview of process mining and delve into three primary categories of process mining techniques, as outlined in Section 5.2.1. Secondly, we offer an insight into Rest web services, with a specific focus on service-oriented and resource-oriented design principles, elaborated in Section 5.2.2.

5.2.1 Process mining methods

Process mining is a field that focuses on the extraction of information related to processes from event logs, thereby transforming them into a structured format [156]. This discipline bridges the gap between conventional process model analysis, often theoretical and lacking

[2] https://drive.google.com/file/d/1AzwmHnxtTx4zZbvAZkDA0dM_UClWtWV1/view?usp=drive_link

real-world data, and data mining, which usually doesn't focus on processes. By visualizing the actual execution of a process based on data, businesses can identify bottlenecks, deviations, and potential for optimization, and hence improve operational efficiency.

Process mining encompasses three major categories of techniques: process discovery, conformance checking, and enhancement. Each of these techniques serves a unique purpose and utilizes different methods and algorithms to carry out their respective tasks. These categories of techniques are detailed in the following sections.

5.2.1.1 Process Discovery

Process discovery is the first category of techniques within process mining. The principal objective is to accurately generate a process model from event logs. The process model serves as a visual and computational representation of the different paths or sequences of activities a case can undergo from its commencement to its conclusion.

The model derived from process discovery is data-driven, formed based on the activity sequence in event logs, and therefore does not make any prior assumptions about the process's structure. The inputs for process discovery are raw event logs obtained from various systems, capturing the progression of various process instances. The output, on the other hand, is a process model that encapsulates the flow of activities, bringing to light aspects such as sequence, parallelism, choices, and loops within a process.

Discovered process models can be broadly classified into three types: procedural, declarative, and hybrid. The procedural models are often used when the control flow between activities is relatively structured and deterministic. They graphically portray the flow and order of activities and often rely on graphical notations such as Petri nets or Business Process Model and Notation (BPMN). Declarative models are used when the process is flexible and the order of activities is not strictly predefined. Instead of specifying a clear sequence of tasks, declarative models focus on the rules and constraints that guide the process. Declarative models can be expressed using languages like Declare. Hybrid models are a blend of both procedural and declarative models.

Various algorithms have been designed to facilitate process discovery, such as the α algorithm, heuristic mining, and fuzzy mining. Each of these algorithms offers different strengths, with variations in complexity, interpretability, and ability to handle noise and concurrency in the event logs. It's important to select the right algorithm based on the characteristics of the specific process and event logs.

5.2.1.2 Conformance checking

The second category, conformance checking, involves comparing a given process model against the real behavior recorded in the event log. This technique checks if reality, as captured in the log, conforms to the model, and vice versa. It helps in detecting deviations, errors, or non-

compliance and can be used to understand why these discrepancies exist. For conformance checking, both an existing process model (which could be derived from process discovery or predefined) and an event log are required. The event log acts as a record of the 'as-executed' reality, while the process model represents the 'as-designed' or 'as-expected' process flow. The output of a conformance check is typically a diagnostic report detailing the discrepancies or deviations between the process model and the event log. This report can highlight:

- Instances where the execution of the process did not adhere to the sequence or flow of activities outlined by the model.
- Cases where the model permits behavior not seen in the real-world execution of the process.
- Activities in the model that never occur in the log, or activities present in the log that the model does not account for.

5.2.1.3 Enhancement and Predictive process monitoring

Finally, the enhancement techniques in process mining aim at improving an existing process model using information about the actual process recorded in the event log. This can be done by extending or repairing the process model. Predictive process monitoring, a type of enhancement technique, focuses on using historical data to make predictions about future process behavior. The goal is to provide timely insights, enabling proactive and corrective actions to enhance process efficiency and reduce potential risks. This can include predictions on the next activities to be performed, remaining completion time, estimated costs, potential rule violations, and more. Predictive process monitoring typically involves two main stages.

1. Create a predictor model: This step aims to build a predictor model that reflects the information learned from historical process data. The raw inputs for this model include historical event logs that record completed instances of the process. These logs contain rich information about the sequence of activities, resources involved, timestamps, and other context data for each case. In addition to the event logs, other sources of information might be included depending on the predictive task at hand. For instance, a process model might be used to provide structural information about the process, while a labeling function might be utilized to define the outcome to predict (e.g., whether a case will violate a rule or the remaining time for case completion). Various machine learning and statistical techniques can be employed to learn the relationship between process variables and the prediction target from this input data. These techniques include, but are not limited to, decision tree, random forests, support vector machines, neural networks, etc. The output of this stage is a predictive model that has learned to capture the normal behavior of a process and can estimate the outcome of interest given the current state of a process instance.

2. Making predictions: The second stage involves the application of the predictive model to ongoing cases to make future predictions. The inputs for this stage are the current and incomplete traces of ongoing process instances. Each trace provides a record of the events that have occurred so far in a case. The predictive model built in the first stage is

then applied to these traces. Based on the learned relationships from historical data and the current state of a case, the model generates an estimate of the future process behavior. For instance, it might predict the next activity in the case, the remaining processing time, the risk of a rule violation, or any other outcome of interest.

5.2.2 Rest APIs: Service-oriented & Resource-oriented

REST is an architectural style for distributed hypermedia systems, which has become the standard for designing web services. A RESTful API (Application Programming Interface) uses HTTP methods to enable communication and data exchange between systems in a stateless manner, meaning each request from a client to a server must contain all the necessary information to understand and respond to the request [108]. RESTful APIs are typically categorized into two main types: Service-Oriented APIs and Resource-Oriented APIs. The primary difference between these two types lies in the organization of their structure and their perspective on data.

1. Service-Oriented REST APIs: Service-Oriented APIs are designed around specific services that deliver operations or functionalities within an application. For example, an API may include a UserService that handles user-related operations, such as creating, retrieving, updating, and deleting users. Each service can encapsulate complex business logic and involve interactions with multiple resources. These APIs typically define endpoints as actions or verbs that correspond to the operations provided by the service. Service-Oriented APIs can use HTTP methods; however, they might not fully exploit the semantics of the HTTP protocol. Often, operations are denoted in the URL itself (e.g., /createUser, /updateUser), and the POST method is frequently employed to execute these operations. For instance, a service-oriented API may have the following endpoints:

- '**POST** /createUser'
- '**POST** /getUser'
- '**POST** /updateUser'

The aforementioned endpoints indicate that the requests are directed to services executing actions, instead of interacting with resources directly.

2. Resource-Oriented REST APIs: Conversely, Resource-Oriented APIs revolve around resources, where a resource is an entity with a type, associated data, relationships to other resources, and methods operating on it. Each endpoint in these APIs corresponds to a distinct type of resource, and the HTTP methods (GET, POST, PUT, DELETE) outline the operations performed on these resources. Resource-oriented APIs aim to use URLs and HTTP verbs semantically, providing an intuitive interface for system interaction. An example of a resource-oriented API structure is:

- '**GET** /users': Retrieve a list of users
- '**GET** /users/id': Retrieve the details of a user

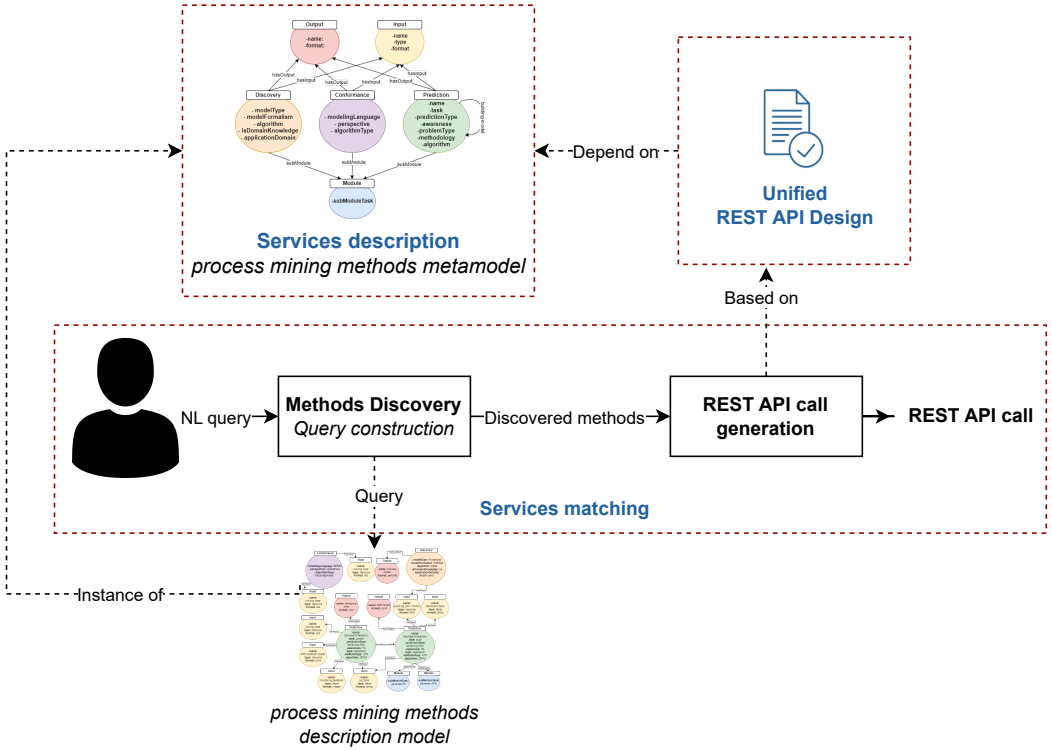


Figure 5.1: Service-oriented architecture overview

- '**POST** /users': Create a new user
- '**PUT** /users/id': Update the details of a user
- '**DELETE** /users/id': Delete a user

The endpoints in this structure show direct interaction with resources (in this case, users) using conventional HTTP methods.

5.3 Overview

Figure. 5.1 shows an overview of the proposed architecture for discovering and accessing process mining techniques. This approach is highlighted by the yellow pointed area in Figure 1.3. The architecture compromises three main components. The first component is services description, which employs a graph metamodel based on LPG to describe the available discovery, conformance, and prediction methods (highlighted by the red pointed area in Figure 1.3). It includes the necessary concepts to represent services, such as the properties of each service, the required inputs and outputs along the potential submodules or micro functionalities that compose the service. The second component is a unified service-oriented REST API design. It leverages the inherent functional nature of process mining methods and provides users with a comprehensive and cohesive service-oriented Rest API design for discovery, conformance, and prediction services. This strategy streamlines the automated creation of REST API calls

for identified instances of these services by seamlessly linking the entities and attributes in the graph metamodel to their appropriate endpoints and parameters within the REST API framework. This design encompasses the definition of URL structure, the judicious selection of HTTP methods, and the meticulous specification of response codes

The final component is services matching, which matches the consumer's requirements with suitable process mining service services. It takes the user's naturally expressed requirements as input and furnishes the corresponding REST API call. Instances of user queries include *'Provide me with available discovery services for generating BPMN model'* or *'Could you recommend prediction services that are capable of predicting remaining time using machine learning models?'*. This component operates through two primary phases. Initially, it initiates by querying the process mining methods graph, which hosts information related to process mining methods, in order to extract all conceivable methods aligning with the user's specifications. This is done by automatically constructing the Cypher query to be executed over the process mining methods graph. To automate this process, we harness the capabilities of large language models like GPT-4 to automatically generate appropriate Cypher queries from the provided NL queries. This is achieved through a meticulous prompt engineering process. These methods are subsequently presented to the user for selection from the available options. Upon the user's selection of an appropriate method, the second stage automatically generates the corresponding REST API call, utilizing the REST API design outlined in the second component.

In the following sections, we describe each of these components in detail.

5.4 Services description: Property graph metamodel

To identify discovery, conformance and prediction services, the necessary properties that distinguish each service as well as the required inputs and outputs should be defined. Therefore, we introduce an LPG graph metamodel that characterizes information related to discovery, conformance and prediction methods. It acts as a foundation for discovering services regardless of the technology employed. The graph metamodel proposed to store process mining methods information is depicted in Figure. 5.2. An example of its instantiation is depicted in Figure. 5.3.

The graph contains six different entity types. Three entity types are used to represent the three main family techniques of process mining. The *Discovery* entity refers to a process discovery method. The *Conformance* entity refers to a conformance-checking method. The *Prediction* entity refers to a prediction method. The inputs and outputs of each method are represented by *Input* and *Output* entities, which are connected to the *Discovery*, *Conformance*, and *Prediction* entities through the relations *hasInput* and *hasOutput* respectively. The *Input* entity has the *name*, *type*, and *format* properties. The *name* property specifies the name of the input. The *type* property specifies the type of input. We distinguish two types of inputs: *literal* and *resource*. *Literal* inputs represent values that are of literal types (e.g. integers, strings, etc.) and must be provided by the user. *Resource* inputs are those that must be

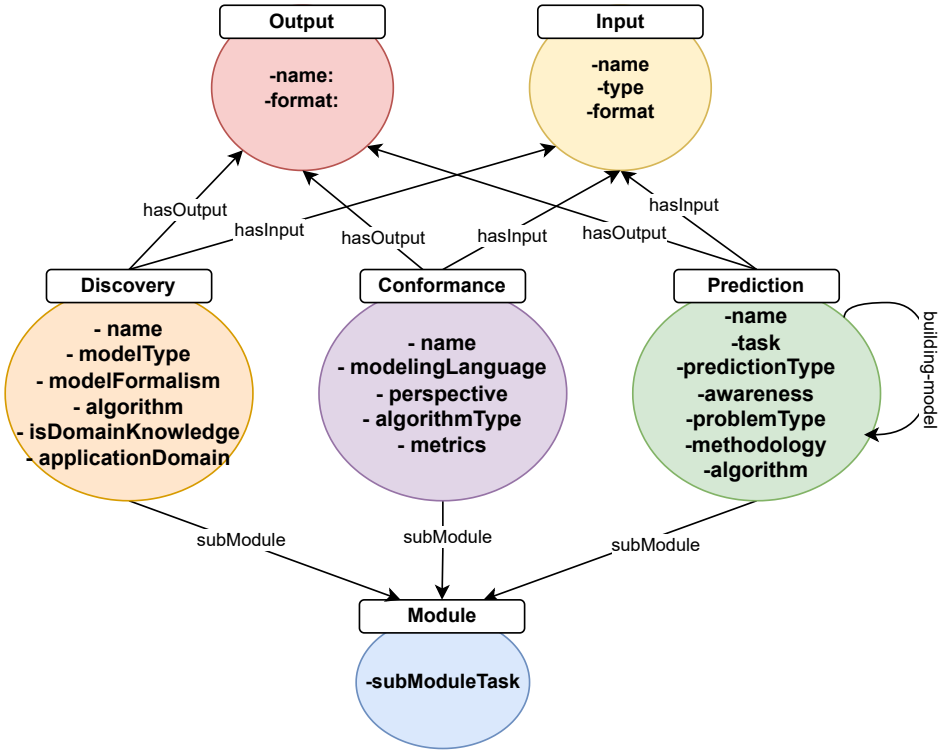


Figure 5.2: Property graph metamodel for process mining methods

obtained from external sources, such as an event log file, a process model, etc. The *format* property specifies the acceptable format of the input, such as XES for event logs, PetriNet for a process model, integers and strings for literal values, etc. Similarly, the *Output* entity has the *name* and the *format* properties.

Moreover, since process mining methods can be complex and may comprise multiple functions (micro functionalities), a *Module* entity is used to represent the functional sub-modules involved in these methods. For instance, consider the Inductive Miner algorithm for process discovery [95], which builds an imperative model typically represented as a Petri Net. This algorithm operates through three key sub-modules: log splitting based on the detection of sequential and parallel activities, creation of sub-logs from these splits, and the final Petri Net model construction from the derived sub-logs. Each of these stages is a functional sub-component of the Inductive Miner algorithm, hence represented as a *Module*.

To connect each method entity with the corresponding *Module* entity, a *submodule* relation is used. Additionally, each Module can have its own inputs and outputs. For instance, as illustrated in Figure. 5.3, the right lower *Prediction* entity represents a prediction method that constructs a Stochastic Petri net (SPN) (specified by the algorithm property) to be used as a predictor model. This method consists of two primary sub-modules: the first one generates a Petri net model, while the second one enriches the Petri net with additional information to create the SPN model. It is important to note that the inputs and outputs of these modules are not included in this example for simplicity.

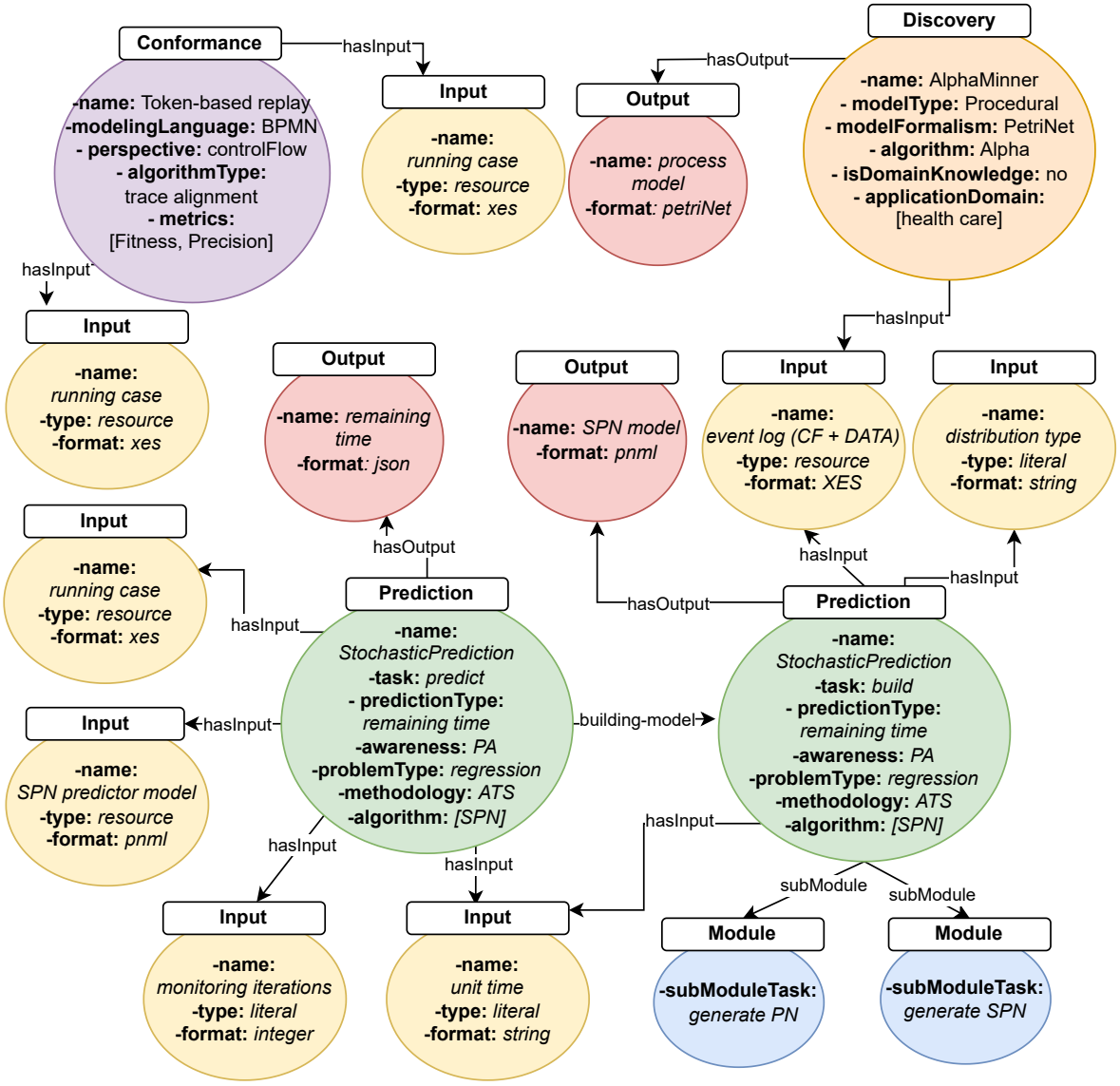


Figure 5.3: Example of property graph model for process mining methods

The *Discovery*, *Conformance*, and *Prediction* entities have properties that characterize these families of techniques. These properties are systematically identified through comprehensive reviews to enable comparison among different techniques within the same family. We have selected the most general and principal properties to characterize these methods. The properties characterizing the discovery, conformance, and prediction methods will be elaborated in the upcoming sections.

5.4.1 Discovery methods properties

The primary properties of process discovery methods are identified through comprehensive systematic reviews [9, 140, 147]. These reviews enable comparison of various existing methods from multiple perspectives. Upon review, we identified six principal properties pertaining to discovery methods. The *name* property defines the method's name. The *modelType* property signifies the class of the discovered process model, which can be categorized into procedural (or imperative), declarative, or hybrid models. Procedural models, also known as imperative models, are characterized by strict sequences of activities depicting clearly defined paths through the process. Conversely, declarative models are founded on the principles of constraints and rules, rather than an explicit sequence of tasks, allowing for more flexibility within the process. Hybrid models combine elements of both procedural and declarative models, accommodating processes that simultaneously necessitate well-ordered activities and rule-bound flexibility. The *modelFormalism* property specifies the specific notation or formalism of the discovered process model. Examples of such formalisms include Petri Nets or Business Process Model and Notation (BPMN) for procedural models, the Declar language for declarative models, etc. The *algorithm* property indicates the algorithm used to discover the model, like the Alpha algorithm or Inductive Miner. The *isDomainKnowledge* property shows whether the method requires domain knowledge, meaning if it needs extra information about the domain, alongside the event log, for the discovery of the process model. Finally, the *applicationDomain* property indicates the application domains where the method has been applied and evaluated. For instance, Figure. 5.3 shows an instance of the Alpha Miner discovery method (i.e. represented by the node with type *Discovery*). This method is used to discover a procedural process model (i.e. the *modelType* property set to Procedural) represented using the PetriNet notation (i.e. the *modelFormalism* property set to PetriNet). Additionally, the method is not domain-specific and has been evaluated in the healthcare domain.

It is important to note that the aforementioned properties are general and can be associated with all discovery methods. However, there are also specific properties that can be employed to categorize and compare certain types of approaches. For instance, the review conducted in [9] reveals that procedural discovery methods, which discover procedural models, can also be compared based on the semantics captured in the model, such as XOR, AND, etc. Regarding hybrid approaches, those that combine existing declarative and procedural process modeling notations, the reviews [147] categorize these methods into three sub-categories: mixed, hierarchical, and parallel, depending on how the procedural and declarative notations are combined to model a process. Furthermore, for methods that incorporate domain

knowledge, the review [140] categorizes these approaches based on the nature of the domain knowledge, whether it's explicit or implicit, and also on the level of interaction with the user, which could range from fully automated to interactive. In the context of this work, we have opted to refrain from diving into such detailed categorizations. Our objective is to maintain a broad perspective, keeping the classification as inclusive as possible.

5.4.2 Conformance methods properties

The main properties of conformance-checking methods were identified from an existing literature review [51] to compare conformance-checking techniques. We identified four different properties. The *name* property defines the method's name. The *modelingLanguage* property denotes the language employed to describe the process behavior, serving as input to the conformance-checking algorithm. The *perspective* specifies the minimum information that the event log file must contain for the conformance method to be executed. For instance, some conformance-checking methods might only require control-flow information contained in the event log file. Conversely, other methods might necessitate supplementary data on various process perspectives, including resources, costs, or duration for a more comprehensive analysis. The *algorithmType* property specifies the algorithm employed to compare the process model to the event log such as log replay and trace alignment. Finally, *metrics* property outlines the set of metrics that are outputted by the method to represent conformance such as Fitness, Precision, Simplicity, and generalization. For instance, Figure. 5.3 shows an instance of a conformance-checking method (i.e. represented by the node with type *Conformance*). This method takes as input a process modeled in BPMN notation (i.e. the *modelingLanguage* property set to BPMN) and an event log representing the control-flow information (i.e. the *perspective* property set to controlFlow). Additionally, it employs the trace alignment algorithm and evaluates conformance using the Fitness and Precision metrics.

5.4.3 Prediction methods properties

The properties of prediction methods are identified by systematic reviews [106, 47] to compare different prediction techniques. We identified seven main properties for prediction methods. The *name* property defines the method's name. The *task* property defines the type of operation that the prediction method performs, which can be either *build-model* or *predict*. Methods that have the *build-model* task create a predictor model using past event log data, such as constructing a machine-learning model to forecast future information. On the other hand, methods that have the *predict* task return a prediction value for an ongoing process based on a built predictor model. The *predictionType* property refers to the aspects of the business process the method predicts, such as the next activity, remaining time, etc. The *awereness* property indicates whether the method is process-aware or not. A method is considered process aware whether it requires an explicit process model as input to build the predictor model. The *problemType* property indicates the type of problem, based on the predicted value, which could be a classification or regression problem. The *methodology* property

indicates the methodology used to build the predictor model, such as machine learning (ML), annotated transition system (ATS), etc. Finally, the *algorithm* property indicates the array of algorithms employed such as decision tree (DT), clustering (CLU), etc.

It is worth noting that, the method for making predictions is normally based on methods for building the predictor model. To capture this relationship, we include a *building-model* relation to the metamodel. For instance, as shown in Figure. 5.3, the left *Prediction* entity predicts the remaining time by utilizing the predictor model constructed by the right *Prediction* entity, which is connected through the *building-model* relation.

5.5 Unified Rest API design

This section introduces the proposed REST API design specifically tailored for invoking discovery, conformance, and prediction services. In this approach, we adopt a service-oriented design to establish a unified interface for invoking these services through a REST API. This choice was motivated by several key factors. First, process mining methods inherently possess a functional nature, emphasizing functionality as a core aspect. Thus, by aligning the API design with the service-oriented paradigm, we provide users with a comprehensive set of process mining services that cater to their specific needs.

Furthermore, our service-oriented design enables the construction of self-descriptive URLs, with endpoints structured around services. This facilitates easy comprehension of the purpose and functionality of the invoked services by examining the URL structure. Consequently, users experience enhanced discoverability and smoother navigation of available process mining services. Moreover, the service-oriented design enables seamless integration of new process mining services and accommodates the evolution of existing ones. This scalability is achieved without requiring significant modifications to existing resource-oriented endpoints, ensuring flexibility and future readiness.

The composition of the API request relies on the elements derived from the graph metamodel presented in Section 5.4 (see Figure. 5.1). This approach facilitates the automatic generation of REST API calls for identified discovery, conformance and prediction services by mapping the entities and properties in the graph metamodel to the corresponding endpoints and parameters of the REST API. The design encompasses the definition of URL structure, selection of HTTP methods, and specification of response codes. Figure. 5.4 illustrates the proposed design showcasing the chosen HTTP methods, URL path, and parameters utilized in the API for the discovery, conformance, and prediction services.

The URL path starts with an endpoint denoting the family of the process mining service-namely discovery, conformance and prediction. This is followed by the specific method name. Subsequently, a collection of path parameters is added, that defines the main properties of the service. These properties are extracted from the *Discovery*, *Conformance*, and *Prediction* entities in the graph metamodel depicted in Figure. 5.2. In the context of *Discovery* services, the URL path embodies the *modelType*, *modelFormalism*, and *algorithm* properties as path

Figure 5.4: Unified REST API design for discovery, conformance and prediction methods

parameters. For *Conformance* services, the URL path integrates the *modelingLanguage*, and *algorithm* properties as path parameters. In the case of *Prediction* services, the URL path includes the *task*, *predictionType*, *methodology*, and *algorithm* properties as path parameters. It is worth noting that while formulating the URL path, we have opted not to include every property displayed in the graph metamodel to avoid prolixity. Instead, we have selectively included key properties which contribute towards making the URL more descriptive, thereby helping users to understand the traits of the service being invoked.

An optional *subModuleTask* parameter can be included in the path, referring to the *subModuleTask* property of the *Module* entity. It allows users to execute a specific functionality of the method by providing the submodule's name. For instance, the following URL *"prediction/GFP/build-model/next-activity/machine-learning/decision-tree"* refers to a prediction service named GFP (i.e. the URL starts with prediction/GFP). The service is responsible for constructing a prediction model (i.e. the task parameter is set to "build-model") that predicts the next activity (i.e. the prediction type parameter is set to "next-activity"). It also specifies that the methodology used is machine learning and the algorithm is decision tree. To invoke a certain functionality of the service, such as log processing, the corresponding module name is appended to the previous URL. This results in a URL like *"prediction/GFP/build-model/next-activity/machine-learning/decision-tree/log-processing"*.

Each path within the API supports the POST and GET operations. Figure. 5.5 illustrates the standardized structure of both POST and GET requests and their corresponding responses. The POST operation handles user input and data required for executing the methods. It generates output result files and stores them on the server. POST request query parameters include literal input values (i.e. *Input* entities with the *type* property set to "literal") and configuration parameter values required to execute the methods and specified by

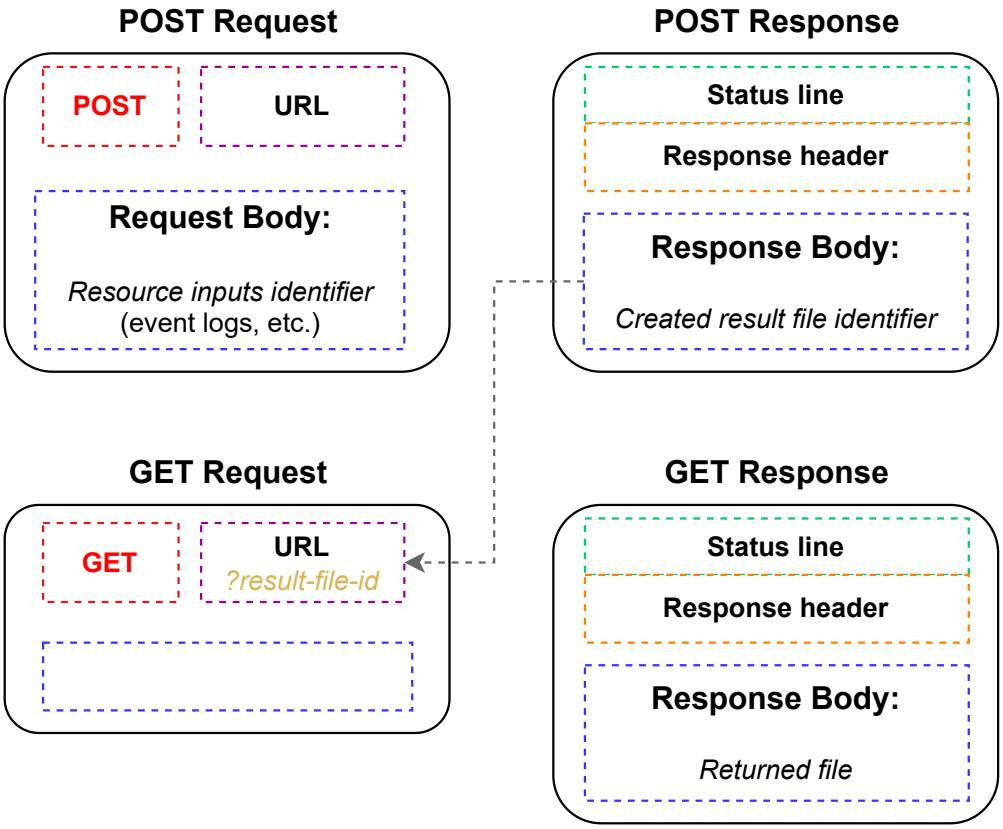


Figure 5.5: General structure of the POST/GET Request and Response

the users. In addition, the POST payload includes resource input values (i.e. *Input* entities with the *type* property set to "resource") that are sourced from external entities, such as event log files, process models, etc. These values are identified using specific identifiers, such as local file paths or URLs. The response body of the POST request includes the identifier of the generated file, which can be used to retrieve the file. By incorporating this information within API responses, clients can dynamically perform the GET operation to retrieve and access the generated file.

The GET method is used to retrieve the created result files. The GET request includes the file identifier as a query parameter, which is used to locate and retrieve the corresponding file from the server. The GET response may vary depending on the specific service being executed and the nature of the returned results. In some cases, the GET response directly presents the result to the user, enabling immediate access and interaction. Alternatively, the result file can be downloaded to the user's device, allowing offline access and further analysis.

5.6 Services matching

This section presents the component responsible for matching user requirements expressed in NL with available services. The matching process involves two stages, depicted in Fig. 5.1. First, the system searches for process mining methods that satisfy the user's requirements by querying the process mining methods description property graph. Then, it automatically generates REST API calls for the selected services, simplifying the process for the user. These stages are described in detail in the following sections.

5.6.1 Stage 1. Method Discovery

This step takes the user's specified requirements as input in NL, along with the process mining methods property graph model. This model is an instantiation of the meta-model described in Section 3.4.2. Then, it identifies the suitable discovery, conformance, or prediction methods that fulfill the specified requirements. An example of this process is illustrated in Fig. 5.6. First, the user specifies the requirements she/he needs to search for in the NL query (step 1 in Fig. 5.6). These requirements may vary from general to increasingly specific. For instance, in the query *'What are the available prediction techniques for the next activity?'*, the user specifies the requirement related to the prediction type of the technique (i.e., predicting the next activity). Conversely, in the query *'Give me available services to predict the next activity using decision tree'*, the user provides more specific conditions concerning the prediction type and the algorithm (decision tree) to be employed.

Next, all available methods that satisfy these requirements are searched by querying the process mining methods property graph using the Cypher query language (step 2 in Fig. 5.6). To automate this process, we harness the capabilities of LLMs like GPT-4 to automatically generate appropriate Cypher queries from the provided NL queries. This is achieved through

a meticulous prompt engineering process, outlined in Figure. 4.3, which effectively guides the language model to produce the desired output.

Our primary objective is to generate Cypher queries automatically from NL queries. These Cypher queries will be utilized to identify process mining services that match the requirements stated in the NL queries. To initiate prompt creation, we provide a concise overview of our service description component, as well as a detailed description of the proposed LPG metamodel, designed to store process mining methods' relevant information. Additionally, we include a task description that informs GPT-4 about the objective of generating Cypher queries for querying the graph.

Similar to the process described in Section 4.8, we continually refine the prompt and evaluate GPT-4's responses. The Perfect Prompt plugin aids in enhancing the provided contextual information and task description. Furthermore, we incorporate examples of property values that could be associated with nodes to assist GPT-4 in correctly mapping values extracted from NL queries to their corresponding graph properties. Additionally, we offer examples of NL queries along with their corresponding Cypher queries to guide the model in producing the desired output. Finally, we instruct GPT-4 not to introduce new node labels or attributes in the generated Cypher queries, restricting the queries to include only information from the described metamodel. This ensures that the generated queries align with the structure and properties defined in the Labeled Property Graph, resulting in more accurate and meaningful query outcomes. Through an iterative approach, we successfully developed a high-quality prompt, enabling effective interactions with GPT-4 to generate accurate and contextually relevant Cypher queries. An illustrative example of this prompt is provided in Appendix A.

Upon generation of the Cypher query, it is subsequently executed over the process mining methods property graph (step 3 in Fig. 5.6). Note that the user can specify conditions for one or more properties of the methods she/he wants to search for. Therefore, the Cypher query is designed to search for all methods that meet the specified properties, even if they have different values for other properties. For instance, if the user is looking for methods that predict the remaining time, the Cypher query will search for all prediction methods that predict the remaining time, regardless of their methodologies or algorithms. As a result, all the identified methods, along with their associated properties, inputs, outputs, and sub-modules, will be presented to the user (step 4 in Fig. 5.6). The user then selects one or many of the discovered methods or sub-modules (step 5 in Fig. 5.6).

5.6.2 Stage 2. Rest API call generation

This step involves automatically generating the associated REST API call for each selected method from the previous step. Before proceeding with the request generation, the user is prompted to provide the necessary values and specify the identifier of required literal and resource inputs through the application console.

The process begins by constructing a POST request that includes all the inputs required to execute the method. Request generation involves constructing the URL path and de-

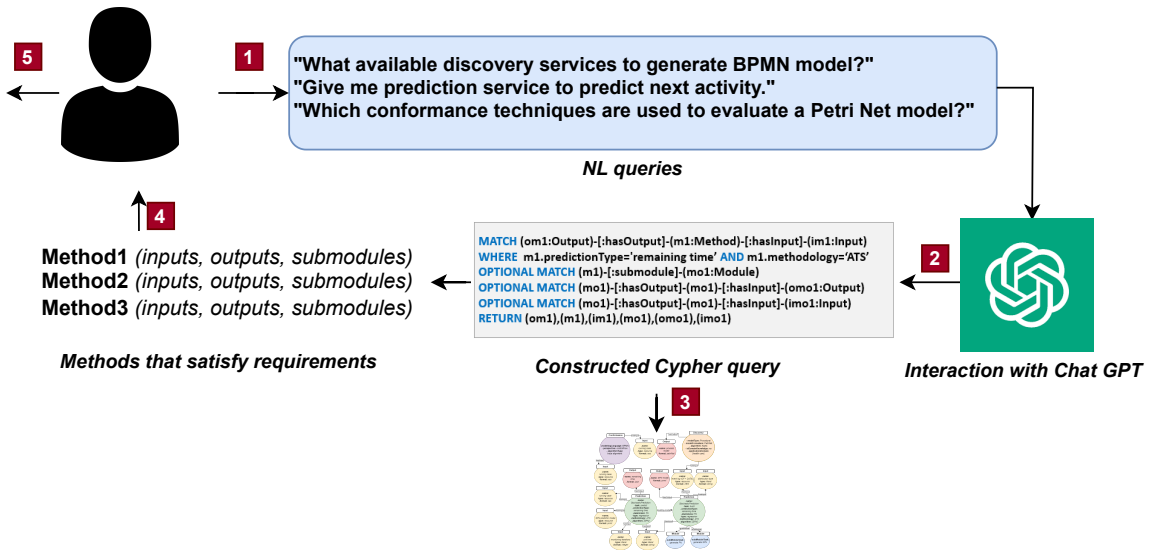


Figure 5.6: Example illustrating the process of discovering discovery, conformance, and prediction methods

termining the possible parameters to be included in both the path and the payload. Algorithm 2 shows the process of POST request generation for each of the selected methods. Initially, the family of the process mining methods (i.e. discovery, conformance, or prediction) as well as the method name is added to the URL path (Lines 3-5). Then, the properties of the selected method, and module if specified, are mapped to the corresponding path parameters, as outlined in Section 5.5 (Lines 6-9). Additionally, all the mandatory literal inputs are included as query parameters, along with the user-provided values (Line 10). For instance, the following URL *"StochasticPrediction/predict/remaining-time/ATS/SPN?monitoring-iteration=...&unit-time=..."* corresponds to the left method entity shown in Fig. 5.3. Next, the locations of resource inputs (e.g. the event log file) are incorporated into the request body payload (Line 12). Once the POST request is created, it is transmitted to the user. The user can initiate the API call using specialized tools like Postman or OpenAI, or directly invoke the REST API through our application. In the latter case, the method is executed and the resulting file is saved on the server. The user receives a POST response containing essential information related to the generated result file such as the file identifier, that can be used to access the generated file. Additionally, the user is also provided with the REST API URL that employs the GET method for retrieving and returning the results. The URL incorporates the generated file identifier as a query parameter, allowing easy access to the desired information.

5.7 Proof of concept

This section provides a demonstration of the proposed architecture through a proof of concept evaluation. The main goals are (i) to evaluate the effectiveness of the service matching

Algorithm 2 POST REQUEST generation

Input: $selectedMethod$, $subModule$, $literalInputs$, $resourceInputs$
Output: $POST\_REQUEST$
$method\_type \leftarrow get\_method\_type(selectedMethod)$
$method\_name \leftarrow get\_method\_name(selectedMethod)$
$URL \leftarrow$ " method_type/method_name/ "
$properties \leftarrow get\_method\_properties(selectedMethod)$
$URL \leftarrow add\_properties\_to\_path\_parameters(URL, properties)$
if *sub_module* is specified **then**
    $URL \leftarrow add\_subModule\_to\_path\_parameters(URL, subModule)$
$URL \leftarrow add\_literalInputs\_to\_quqery\_parameters(URL, literalInputs)$
$POST\_REQUEST \leftarrow URL$
$POST\_REQUEST \leftarrow add\_resourceInputs\_to\_payload(POST\_REQUEST, resourceInputs)$

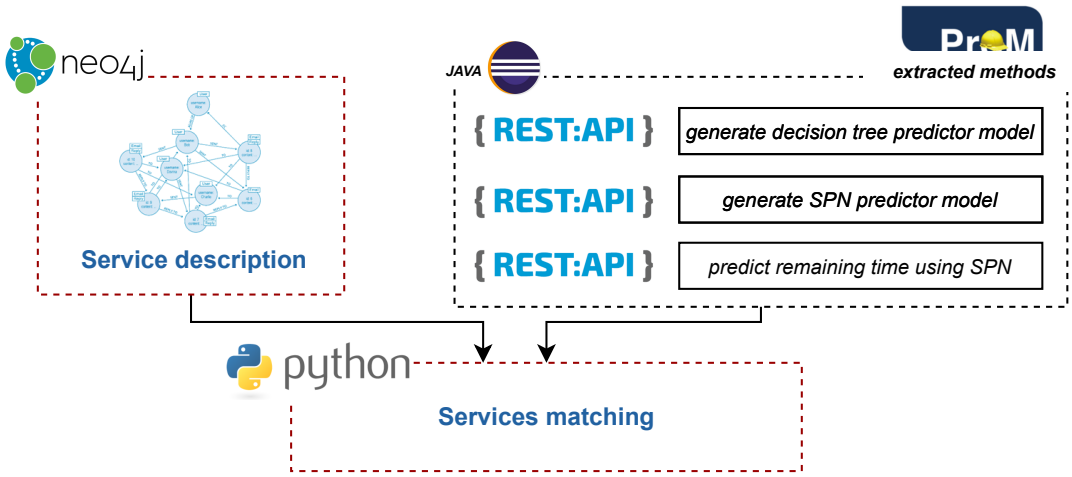


Figure 5.7: Technologies employed in developing the proposed Architecture

component to identify suitable process mining methods based on user queries expressed in NL, (ii) to evaluate the accuracy of the service matching component in generating REST API calls and (iii) to demonstrate the practical feasibility of implementing REST APIs for existing methods within the context of process mining. Figure. 5.7 showcases an overview of the used technologies for developing and evaluating the proposed architecture. A set of prediction methods, meticulously chosen from the existing literature, was employed to populate the property graph dedicated to process mining methods. This data is stored in Neo4j. The service matching component is developed as an independent Python application. This application interacts with users by accepting NL requests for process mining service discovery. It connects to the GPT-4 API using an API key, transmitting a predefined prompt to guide the model in constructing appropriate Cypher queries and the user's request. Once the constructed Cypher query is received, it is then executed over the Neo4j property graph to retrieve the relevant discovered services. The user is then presented with the list of discovered services, complete with their corresponding inputs, outputs, and submodules. Upon selecting

	Discovery	**Conformance**	**Prediction**
Queries generated manually	18	8	9
Queries generated using Quillbot	9	5	6
Queries generated by GPT-4	21	20	20
Total number of queries	42	33	35

Table 5.1: Number of NL queries generated manually, using Quillbot tool and GPT-4 related to discovery, conformance and prediction services

desired services, the application generates the corresponding REST API details, including the URL, parameters, and HTTP method. In addition, to evaluate the Rest API implementation and call, three distinct REST APIs were implemented. These APIs were designed to provide access to three distinct prediction methods, all of which are available within the ProM tool. The APIs were implemented using the Java programming language.

The execution of the proof of concept entails two primary experiments. The first, detailed in Section 5.7.1, is dedicated to evaluating the effectiveness of the process mining method discovery from NL queries. The second experiment detailed in Section 5.7.2, is dedicated to evaluating the REST API call generation and the feasibility and real-world applicability of REST API implementation for process mining methods. The complete source code, essential files, and comprehensive instructions are available at `https://github.com/merianakb/SOA.git`.

5.7.1 Methods discovery evaluation

This section presents the experiment performed to evaluate the effectiveness of the service matching component to identify suitable process mining methods based on user queries expressed in NL. In other words, it assesses how well the large language model, namely GPT-4, could create accurate Cypher queries to address NL queries after the prompt definition explained in Section 5.6.1. To conduct the evaluation, a total of 110 NL requests were generated to discover services related to processes of discovery, conformance, and prediction. These NL requests were created using three methods: (i) manual generation, (ii) employing Quillbot, a paraphrasing tool[3], and (iii) leveraging GPT-4 as a language model. The breakdown of NL queries created through manual methods, Quillbot, and GPT-4, all pertaining to discovery, conformance, and prediction services, can be found in Table 5.1. A selection of examples showcasing these NL queries is presented in Table 5.2.

The generated queries were divided into two categories. The first category (*G1*) encompassed queries seeking services without any specific conditions on the properties of the service, or with simple conditions tied to a single property. For instance, examples of *G1* category queries include: *'Which prediction services can anticipate the next activity in an ongoing process?', 'What available discovery services are there?', 'Can you describe trace alignment-based*

[3]https://quillbot.com/

	NL queries
Discovery	There are any process discovery techniques that produce BPMN model using the alpha minner algorithm?
	Can you list discovery methods that generate declarative models using the Declare formalism?
	I'm looking for hybrid models that have been applied in the e-commerce domain. Can you list them?
	Are there any discovery methods specialized for the education sector that provide declarative models?
	Which discovery services generate models in either BPMN OR PetriNet formalism?
Conformance Checking	Can you provide accessible conformance techniques between a BPMN model and multi-perspective event log
	Are there any methods that use the Declare language and either output Fitness metrics OR generalization metrics?
	I'm searching for techniques that either employ PetriNet OR BPMN language for process behavior description. Any suggestions?
	Which conformance-checking techniques are designed for a control flow perspective of the event log and provide Simplicity metrics?
	Can you recommend conformance-checking methods that focus on the control flow perspective and either employ PetriNet OR Declare language for process behavior?
Prediction	Provide me with services for remaining time prediction using an annotated transition system or Naive Bayes algorithm
	Are there prediction services that focus on outcome forecasting and use the decision tree algorithm?
	I'm searching for services that are not process-aware and employ statistical methodologies to predict LTL violations. Any recommendations?
	Which prediction services use either machine learning OR annotated transition systems to forecast the next activity?
	Which prediction methods use the decision tree algorithm and are designed to forecast LTL violations?

Table 5.2: Example of NL queries used to evaluate the matching services component

conformance testing techniques?', etc. The second category (*G2*) included queries aimed at finding services by imposing complex conditions using logical AND/OR combinations of different values related to either the same or different properties. Examples of *G2* category queries include: *'Are there any process discovery techniques that utilize the alpha miner algorithm to produce BPMN models?', 'Which discovery services generate models in either the BPMN OR PetriNet formalism?', 'Which prediction services use either machine learning OR annotated transition systems to anticipate the next activity?', 'Could you list prediction methods using the decision tree algorithm designed for forecasting LTL violations?', etc.*

To assess the constructed Cypher queries, two evaluation metrics were established. The first metric is the accuracy of correctly constructed Cypher queries from a syntactical perspective. It calculated the ratio of accurately constructed syntactically valid Cypher queries to the total number of requests made. The second metric is the accuracy of semantically correct Cypher queries. This metric determined the ratio of appropriately formulated Cypher queries that were semantically correct – that is, queries yielding relevant services aligned with user needs – to the total number of requests. For checking syntactic correctness, the Neo4j tool was employed, while the assessment of semantic correctness was conducted manually to ensure that the queries produced relevant outcomes according to user requirements.

Results & Discussion

The evaluation results of the categorized requests are shown in Table 5.3. The first column showcases the symbols representing each category, inclusive of the average row. The second

	Accuracy of syntactically correct Cypher queries	Accuracy of semantically correct Cypher queries
G1	1	0.94
G2	1	0.98
Average	**1**	**0.96**

Table 5.3: Accuracy of syntactically and semantically correct constructed Cypher queries for categories G1 and G2 with the overall average

column displays the accuracy of correctly constructed Cypher queries from a syntactical perspective. The third column displays the accuracy of semantically correct Cypher queries. A perfect accuracy score of 1 was achieved for syntactically formulated Cypher queries across both *G1* and *G2* categories. This outcome underscores GPT-4's adeptness in generating syntactically accurate Cypher queries. GPT-4 demonstrated its capability to construct Cypher queries of varying complexity, incorporating different combinations of conditions related to service properties. Furthermore, when a service property involves an array of values – like the 'algorithm' property for prediction services or the 'metrics' property for conformance services – the constructed Cypher queries efficiently search for the specific value(s) specified by the user within that array.

In terms of the accuracy of semantically correct Cypher queries, a commendable score of 0.96 was attained. This outcome indicates that the prompt created through the prompt engineering process was of high quality, guiding GPT-4 effectively in generating accurate Cypher queries. For both simple and complex user queries, GPT-4 was successful in constructing appropriate Cypher queries that yielded the requested services. On analyzing the incorrectly formulated queries, a prominent issue emerged. It was observed that certain semantics within user requests proved challenging for GPT-4 to interpret accurately. For instance, in the user query *'Please provide me with discovery strategies that do not require user interaction'*, GPT-4 struggled to interpret the phrase 'do not require user interaction' as a property of the discovery service, specifically the 'domain knowledge' property set to 'No'. Similarly, for user queries like *'I require LTL violation prediction services that do not require a process model as input and produce a decision tree as a predictor model'*, GPT-4 faced difficulty in mapping the phrase 'do not require a process model' to the 'NPA' value representing the awareness property of the prediction service. Furthermore, in the user query *'What services are available to check the consistency of a process model and an event log?'*, GPT-4 interpreted 'consistency' as a value of the 'metrics' property for conformance services. These instances of inaccuracy mainly stemmed from the complexity of certain semantics that posed challenges for GPT-4's interpretation.

In conclusion, the evaluation highlighted GPT-4's consistent proficiency in constructing both syntactically and semantically accurate Cypher queries. The model demonstrated its competence in formulating queries with precision across both simple and complex query categories (*G1* and *G2*). However, while achieving success in query accuracy, challenges emerged in handling nuanced semantics and complex logical conditions. These instances led to occasional inaccuracies in the construction of semantically accurate queries.

5.7.2 Rest API generation evaluation

This section presents the conducted experiments with the dual purpose of (i) evaluating the accuracy of the service matching component in generating REST API calls and (ii) demonstrating the practical feasibility of implementing REST APIs for existing methods within the context of process mining. To assess the precision of REST API call generation, the implemented Python application designed for method discovery through NL queries (referred to as the methods discovery step) was utilized. From the available pool of discoverable methods, along with their associated properties, a specific subset of methods was chosen. The Python application was then tasked with generating the corresponding REST API calls for these selected methods. This process involved producing the required URL, parameters, and HTTP method for each method within the chosen subset.

For the evaluation of the feasibility and practicality of implementing REST APIs for process mining methods, three prediction methods were selected from the existing literature, specifically from references [40] and [133]. These methods had been incorporated as plugins within the ProM tool. The first method, described in [40], involved generating decision trees from event logs to function as prediction models. These decision trees could forecast outcomes, next activities, or final attribute values. The second and third methods discussed in [133], centered around constructing Stochastic Petri-nets (SPNs) from event logs and subsequently utilizing these SPNs to predict the remaining time of an ongoing process. The process of building the SPN model consisted of two primary steps: creating a Petri-net from the event log and then enhancing the Petri-net with supplementary data to obtain an SPN. This model building was viewed as comprising two sub-modules of the overarching building method.

To realize the corresponding REST APIs, the Java source code of the two implemented packages in Eclipse was accessed. Appropriate modifications and adjustments were made to the code before implementing the REST APIs using the Java programming language. Consequently, three REST APIs were developed: one for building the decision tree as proposed in [40], one for constructing the SPN model as described in [133], and a third for querying the constructed SPN model to retrieve prediction values. These implemented REST APIs were then invoked from the Python application as part of the experimental assessment.

Results & Discussion

The empirical outcomes of our evaluation reveal that the service matching component exhibited remarkable precision in generating REST API calls for each of the selected methods. This accuracy extended to both the URL construction and the parameter specification, adhering to the requirements for each individual process mining method. The ability to generate such precise API calls suggests a high level of accuracy in our service-matching component, underscoring its reliability and effectiveness for practical applications. This result also confirms the system's capability to bridge the gap between user-generated NL queries and the technical specifications needed to invoke process mining methods.

Moreover, the successful implementation of REST APIs for existing process mining meth-

Evaluation criteria	Description
Clarity and Readability	The extent to which the REST API is easy to understand so that users can easily determine the services and resources they are accessing.
Relevance	The extent to which the REST API accurately reflects the properties of the service being used.
Coverage	The extent to which the REST API design effectively represents the diverse range of services and their distinct characteristics.

Table 5.4: Evaluation criteria of the REST API design

ods serves as empirical evidence for the feasibility and applicability of the proposed architecture. This not only demonstrates the utility of our approach but also sets a precedent for how extant computational methods in process mining can be made more accessible and integrable through RESTful services. However, a notable limitation of this approach was the time-consuming nature of the REST API implementation process. A significant portion of this time was spent on understanding the original Java source code of the selected methods. Subsequent to that, additional time was invested in making necessary modifications and invoking appropriate functions. Although it was a laborious task, it was observed that many of the changes needed were common across the three different methods. This observation suggests that there may be an opportunity to streamline this process through more generalized solutions or templates, potentially cutting down on the time and complexity involved in future implementations.

Given these findings, we strongly encourage developers in the process mining field to consider providing REST API accessibility for their proposed methods. Doing so would significantly ease the integration and utilization of these advanced techniques into various systems and platforms, thereby amplifying their reach and impact.

5.8 Evaluation

This section presents the conducted experiments aimed at evaluating user experiences and the effectiveness of the proposed REST API design. The qualitative evaluation of the proposed REST API design is based on specific criteria outlined in Table 5.4, including clarity and readability, relevance, and coverage. These criteria are selected due to their significance in ensuring the overall quality of the service-oriented API design. In this evaluation, our primary focus has been on the prediction methods, which are inherently the most complex techniques. Consequently, we anticipate the same outcomes when applying the evaluation to both discovery and conformance techniques. In the first experiment, we assess the user experience in searching for and accessing prediction methods using traditional methods. Additionally, we evaluate the clarity and comprehensibility of the proposed REST API design (Section 5.8.1). In the second experiment, we focus on evaluating the coverage and relevance of the REST API design (Section 5.8.2). All evaluation results and necessary files are available at `https://github.com/merianakb/SOA.git`.

5.8.1 Use case

A use case was conducted with external users, with two main goals. First, the aim was to assess the user experience when searching for and accessing prediction methods using conventional methods such as process mining tools or literature searches. This use case focused on identifying the challenges faced by users while attempting to apply prediction methods in practice, emphasizing the necessity for an automated solution to discover and access these methods. Second, the use case sought to evaluate the clarity and readability of the proposed Rest API. Section 5.8.1.1 outlines the methodology followed in the use case, including participant selection and different experiments conducted. The findings of the study are discussed in Section 5.8.1.2.

5.8.1.1 Methodology

The use case involved participants from diverse backgrounds and expertise, including data scientists, software engineer students, Ph.D. students, and developers. All participants had a solid understanding of process and data analysis. However, their familiarity with REST APIs varied, ranging from extensive knowledge to basic understanding. Some participants had prior experience with process mining tools like ProM, while others were new to such tools.

An overview of process mining, prediction techniques, and the ProM tool was first presented to the participants. The study was divided into three parts. The first part aimed to assess the participants' experiences with searching for and accessing prediction methods using the ProM tool. Participants were asked to install the latest version of the ProM Nightly build[4] and were given access to documentation[5] of all available packages in ProM. To access a particular plugin in ProM, users must locate and install the package that contains the plugin in ProM's package manager. In this use case, participants were asked to find prediction packages, install them, and launch the required plugins to execute one prediction method. It should be noted that there were no limitations imposed on the prediction methods that could be searched for or executed during this initial phase of the study. In the end, we requested participants to give their feedback[6], on the challenges they encountered while searching for the prediction methods using one of the process mining tools and whether they were able to obtain the required information about each identified plugin.

The second part aimed to evaluate the user experience of searching for prediction methods with specific requirements using the literature. To support the search process, participants were given two tables of existing prediction methods from recent systematic reviews [106, 47]. Eight different use cases for searching a specific prediction method were then presented. For each use case, participants were asked to search for an existing published article that had proposed a prediction method that meets the requirements and has been made available.

[4]https://promtools.org/prom-6-nightly-builds/
[5]https://svn.win.tue.nl/repos/prom/Packages/
[6]https://forms.gle/3ruRmnDDYowAmFJe6

Following their search, the participants were required to complete a Google Form[7] containing the list of articles discovered for the use cases as well as the duration spent on the search.

Finally, the third part aimed to evaluate the participants' experience of searching for prediction methods that met specific requirements by using REST API. Through this evaluation, we also aimed to assess the clarity and readability of the proposed URL design. To achieve this, the participants were provided with 15 different REST APIs for existing prediction techniques, which were formatted according to the design outlined in Section 5.5. Similar to the second part, they were tasked with searching for the REST API that met the requirements described in each of the eight use cases. In addition, they were required to submit their findings via a Google Form[8], which included a series of questions about the identified REST APIs, the time it took to complete the task, their feedback on the REST API design, and their preference for using REST APIs or a process mining tool.

It is important to highlight that the workshop was conducted in multiple sessions, not all at once, to accommodate the participants. The workshop took place three times, with one session held in person at the university and two sessions conducted remotely. The same procedure was followed in each workshop, covering all three parts of the workshop. Participants were given the flexibility to choose the order in which they performed the workshop parts based on their preferences. Additionally, there was no strict time constraint imposed on the participants. They were free to complete the workshop parts and the corresponding Google Forms at their own pace, although we requested an approximate completion time for reference purposes. As a result, 33% of the participants opted to begin with ProM, then proceed to literature search, and finally utilize the REST APIs. 58% of the participants started with ProM, followed by using the REST API and then conducting a literature search. Lastly, 9% of the participants chose to start with the REST API, then perform a literature search, and conclude with the utilization of the ProM tool.

5.8.1.2 Results & Discussion

The evaluation results collected from the participants regarding their experiences in searching prediction methods using ProM indicate that all users were able to find at least one prediction plugin. However, the specificity of each plugin and the properties of the executed methods were not always clear to the participants. Specifically, 41% of the participants were able to identify only the type of information the method is able to predict, while 50% of participants were unable to determine any properties of the found prediction plugins. The lack of the necessary documentation, and the ambiguity of each plugin's information, were the main challenges faced by all of the participants.

Regarding user experience with searching for prediction methods using the literature, despite providing the participants with comparative tables of all the properties of existing methods extracted from systematic reviews, only 47% of the responses met the requirements.

[7] https://forms.gle/jGnd7ZBS2xRA3cem8
[8] https://forms.gle/vNQn7Td8R7yxhmWz7

Most of the correct responses were provided by Ph.D. students who were already familiar with scientific articles. We also observed that data scientists, developers, and engineering students completed the task in less than 30 minutes, but they did not provide accurate responses. As a result, the use of literature and published articles for searching prediction methods is a challenging task, especially for data and process analytics.

Finally, regarding the user experience and feedback on the REST APIs, the results indicate that the proposed REST API design successfully met the objectives of clarity and readability, achieving an 89% accuracy in correctly associating the REST API with the corresponding requirements. This design effectively communicates the characteristics of the prediction service, enabling users to comprehend its specific features better and choose the appropriate REST API that aligns with their needs. Furthermore, participants provided positive feedback about the REST API design. They praised its clarity, mentioning that it effectively specified the type of prediction, problem, algorithm, etc. The design was also appreciated for its simplicity, as participants found it intuitive to understand the desired action, available inputs, and method just from the URL query.

Interestingly, the order in which participants completed the three parts did not have a significant impact on the results. Even participants who first engaged with the Rest APIs before conducting the literature search demonstrated comparable understanding, suggesting that there was no learning effect in the early phase that influenced the comprehensibility of the Rest APIs.

Furthermore, the participants provided overwhelmingly positive feedback about the REST API design when asked for their opinions. For instance, several participants praised the design's clarity, with responses such as "*the method is made clear by specifying the type of prediction, type of problem, method, predictor model, and produced model*", and "*it is more intuitive to use a formalism with REST APIs. Standardization can attract more audience*". Other participants appreciated the design's simplicity, with one saying "*just from the URL query, we can understand what we want to do, what inputs we have, and the method to use*".

When asked for their preference for using REST APIs or a process mining tool, all participants preferred REST APIs. They provided several reasons for this preference, including the freedom that REST APIs provide to users. One participant explained, "*This gives more freedom to users. They don't have to install ProM on their machines. Calling an API endpoint is much easier than launching a plugin*". Another participant preferred REST APIs because they offer a guided approach, saying, "*The REST approach starts with the results, and we show the itinerary the user takes. So I think being guided is better than looking for information from scratch*". Participants also appreciated the intuitive logic of REST APIs, with one stating, "*It has an intuitive logic. Just follow the parameters written in the REST*".

The results of our study demonstrated the difficulties that users encounter when searching for and accessing prediction methods through traditional methods such as using a tool or searching through literature. Moreover, the participants' feedback and preferences indicate that the REST API design was well-received and considered a more user-friendly option than a process mining tool.

5.8.2 Evaluation of coverage and relevance of Rest API design

In this section, we present the experimental evaluation conducted to assess the coverage and relevance of the service-oriented Rest API design. Specifically, Section 5.8.2.1 outlines the methodology employed in the evaluation while the results are discussed in Section 5.8.2.2.

5.8.2.1 Methodology

The evaluation of the REST API's relevance and coverage was conducted by the authors using a set of 12 diverse published articles as a foundation. These articles encompassed a wide range of prediction methods with distinct properties, including the construction of predictor models, prediction of specific information, or a combination of both. Through a systematic analysis, the relevant properties of each method were identified, focusing on the *Prediction* entity depicted in Figure. 5.2. The inputs, outputs, and potential sub-modules associated with each method were extracted based on the descriptions provided in the articles, without delving into technical details.

Once the pertinent information for each method and sub-module was identified and extracted, the next step involved defining the corresponding REST API URLs following the unified design outlined in Section 5.5. The properties of the prediction methods proposed in each article, such as prediction type, methodology, problem type, and algorithm, are presented in Table 5.5. The rows in the table represent the properties, while the columns indicate the article IDs.

5.8.2.2 Results & Discussion

The evaluation results indicate that the proposed REST API design effectively represents various prediction methods along with their inputs, outputs, and properties. However, a significant limitation was identified for methods that had multiple values assigned to the same property. This was particularly observed in the methods proposed in the highlighted articles in Table 5.5 (i.e. [40, 46, 96, 152, 159]). First, some methods proposed prediction models that could predict multiple types of information (e.g. the method proposed in [40]). Nonetheless, the proposed REST API design only accounts for one prediction type in the path. Consequently, when dealing with methods that have multiple prediction types, the creation of several REST APIs is necessary, with each API corresponding to a distinct prediction type. Moreover, some methods may use multiple algorithms (e.g. methods proposed in [46, 96]) or involve both classification and regression problems (e.g. the method proposed in [159]). However, the proposed REST API design can only include one value for each property. As a result, one of the values must be chosen to be represented in the REST API, which may not capture the full range of method properties.

In conclusion, our evaluation demonstrates that the proposed REST API design success-

Reference	Prediction Type	Methodology	Problem Type	Algorithm
[40]	next activity last value outcome	ML	CLASS	DT
[133]	time	ATS	REG	SPN
[46]	LTL	ML	CLASS	DT CLU
[119]	next activity time	ML STAT	CLASS REG	SVR NB
[39]	risk	similarity	CLASS	RP
[33]	next activity	STAT	CLASS	EM
[96]	outcome	ML	CLASS	HMM DT
[118]	risk	STAT	CLASS	RP
[32]	next activity	STAT	CLASS	EM
[152]	next activity time	ANN	CLASS REG	LSTM
[105]	indicator	STAT	CLASS	evolutional
[159]	outcome time	ML	CLASS REG	SVM

Table 5.5: Properties of prediction methods proposed in the articles

fully covered a wide range of prediction methods and their properties, meeting the coverage criterion. However, in terms of relevance, the design accurately represented the properties of most methods, with the exception of those that have multiple values for the same property. In such cases, the design was unable to accommodate all the values, potentially affecting the relevance of the REST API design for those specific services.

5.9 Conclusion

This chapter introduces an innovative approach to tackle the challenges associated with discovering and accessing process mining methods. This approach is based on a service-oriented architecture that places a strong emphasis on REST APIs. The proposed architecture consists of three key components: (i) service description component responsible for representing the properties of discovery, conformance, and prediction services using an LPG; (ii) unified service-oriented REST API design which establishes a unified design for process mining methods through REST APIs, making it easier for users to access and utilize these methods; and (iii) a services matching component responsible for matching user requirements expressed in NL to services. It leverages the capabilities of the LLM GPT-4 to automatically construct Cypher queries from NL. These queries are executed over the process mining methods property graph to retrieve the corresponding methods. Subsequently, a REST API call is generated based on the user's selected service.

Our evaluation primarily focused on prediction methods, known for their inherent complexity. We anticipate that similar results would be obtained when applying the same evaluation process to both discovery and conformance techniques. The results from our experiments were highly promising and provided the following insights:

- The proof of concept results confirmed the system's ability to bridge the gap between user-generated NL queries and the technical specifications required to invoke process mining methods;
- GPT-4 consistently demonstrated its proficiency in constructing both syntactically and semantically accurate Cypher queries. However, handling nuanced semantics and complex logical conditions posed occasional challenges, leading to minor inaccuracies in query construction;
- The service matching component demonstrated remarkable precision in generating REST API calls for selected methods, including URL construction and parameter specification, meeting individual process mining method requirements;
- Users faced difficulties when searching for and accessing process mining methods through traditional methods or literature. The REST API design was well-received as a more user-friendly alternative to a dedicated process mining tool;
- The developed REST API design was user-friendly, adhering to clarity, readability,

and coverage criteria. However, it occasionally struggled with methods having multiple values for the same property;

In light of these findings, we strongly encourage developers in the process mining field to consider offering REST API accessibility for their methods. This approach would greatly simplify the integration and utilization of advanced techniques across various systems and platforms, expanding their reach and impact.

While our current approach emphasizes a service-oriented REST API design focused on functional aspects, future work could explore a resource-oriented design paradigm. This alternative approach would restructure the API design to organize resources around the diverse outputs generated by process mining methods, rather than solely emphasizing functionality. This restructuring would involve identifying and defining these outputs as distinct resources and adapting the URL schema accordingly. The choice between service-oriented and resource-oriented designs should be made after thorough investigations and consultations with users and organizations. Additionally, there is potential to extend the solution by introducing a service composition formalism, enabling users to create new services using pre-existing ones. The next chapter delves beyond the scope of this thesis, proposing extensions and new perspectives for each of the contributions presented herein.

Chapter 6

Expanding Horizons: Envisioning Extensions and Refinements

Contents

6.1 Introduction

In the dynamic realm of process analysis, continuous evolution and refinement are not just commendable, but often essential. As this thesis has unveiled three pioneering contributions toward conversational AI frameworks for cognitive process analysis, it is both prudent and intriguing to cast an exploratory gaze into the horizon, considering potential avenues for their extension and refinement. Such explorations can not only amplify the efficacy of the initial contributions but also pave the way for groundbreaking advances in the future. The primary aim of this chapter is to shed light on those areas of enhancement and expansion that lie just beyond the current ambit of our work. These potential pathways, while based on the bedrock of our initial research, reach out further, striving to encapsulate a broader spectrum of possibilities and address a wider range of challenges in the realm of process analysis.

We delve into a thorough examination of each of our contributions, actively seeking opportunities for enhancement and expansion:

- **Graph Meta-Model for Storing Process Execution Data:** Within the complex web of process execution data, our graph meta-model stands as a meticulous custodian. Yet, its true potential lies not just in its current form, but in its adaptability. How might it evolve to more comprehensively encapsulate process intricacies? How might its boundaries expand to seamlessly integrate both instance and process-level information?
- **Natural Language Querying of Process Execution Data:** Bridging the chasm between human linguistic patterns and structured database queries is no small feat. Yet, the landscape of natural language is vast, and the horizon of our interface can stretch even further. What other NL queries might it capture? How might it adapt to interact with a diverse array of meta-models, expanding its repertoire of interpretative capabilities?
- **Service-Oriented Architecture for Process Mining Techniques:** Our architectural contribution is robust, but the dynamism of process mining demands a structure that is both sturdy and flexible. Where can we introduce more layers of granularity? How can its design be enhanced to better mirror the rapidly shifting terrain of process mining?

In the rest of this chapter, each section unfolds as a dedicated exploration of one of our pivotal contributions. Section 6.2 provides an in-depth look into the potential enhancements of the graph metamodel for storing process execution data. In Section 6.3, we transition into the vast domain of natural language querying, pondering its broader capabilities and adaptability. In Section 6.4 we delve into the architectural nuances of our service-oriented approach, scrutinizing its design elements and proposing visionary refinements. Finally, Section 6.5 concludes the chapter.

6.2 Extending the Graph Meta-Model for Storing Process Execution Data

In Chapter 3, we introduced a graph metamodel based on LPG to provide a comprehensive and explicit representation of all aspects inherent to multi-dimensional process data. With a sharp focus on the details of instances, it effectively captures a wealth of information, including executed activities, involved actors, impacted data and objects, and the intricate relationships between these entities. Notably, we underscored the importance of harmonizing both structural and behavioral data within this metamodel.

Unlike many other database technologies, graph databases offer remarkable flexibility, allowing for the addition of new elements without undue complexity. This presents an opportunity to enhance the current metamodel, making it more comprehensive and information-rich. In the forthcoming sections, we take a step beyond the current metamodel version and

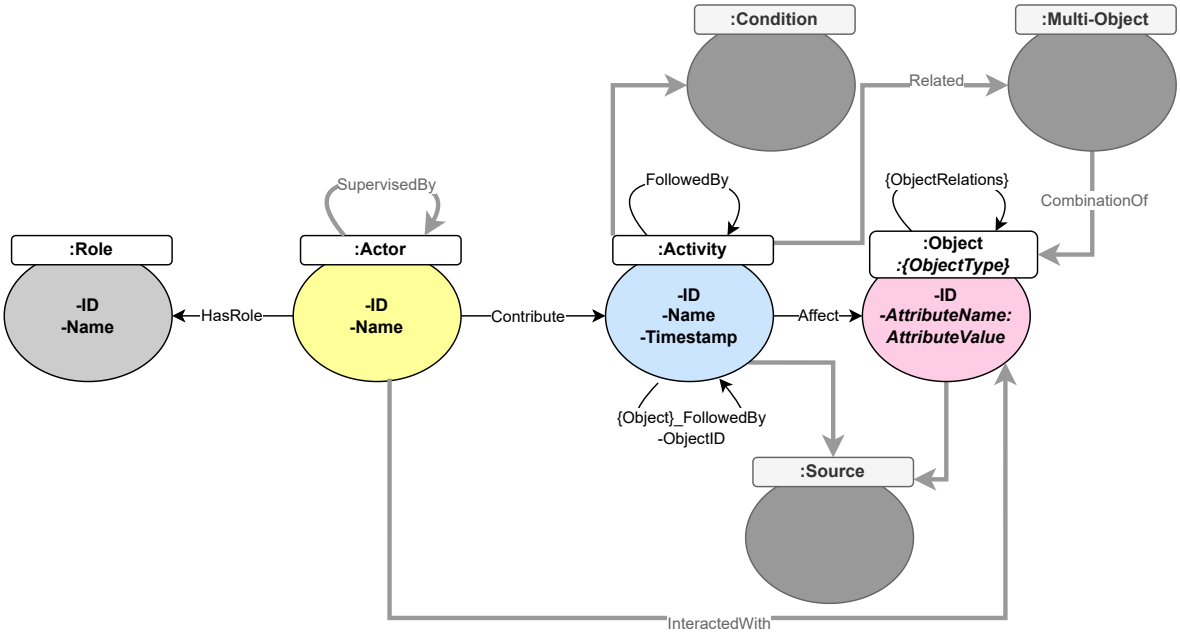


Figure 6.1: Examples of new nodes and relations incorporated into the graph metamodel for instance-level enhancement

explore avenues for its extension. We delineate two distinct levels of exploration. The first pertains to augmenting the graph metamodel with additional instance-level information, as detailed in Section 6.2.1. The second delves deeper into the expansion of the graph meta-model, encompassing not only instance-level data but also the representation of process-level information, which can be instrumental in deriving insights at the process level, as expounded in Section 6.2.2.

6.2.1 Instance-level extensions

Instance-level enhancements are centered around the goal of fine-tuning the graph meta-model to encompass a wider array of elements, specifically designed to enrich the information related to process execution data. The driving force behind these enhancements is to ensure that the meta-model accurately captures a more nuanced and comprehensive portrayal of individual process instances. This refinement can take various forms, including the introduction of diverse node types, the fine-tuning of relationships between nodes, or the augmentation of node attributes. Figure 6.1, illustrates potential additions to the graph meta-model, showcasing new nodes and relationships that could be integrated. These enhancements aim to enrich the model's ability to offer deeper insights into the details of process execution data. The following elaboration provides a more detailed insight into potential instance-level enhancements to the meta-model:

1. **Inclusion of additional node types**: This involves the incorporation of new types of nodes within the graph meta-model to encapsulate more specific or composite aspects

of a process. For instance:

- *Multi-object Nodes*: These nodes symbolize intricate scenarios where multiple objects partake in a singular activity. For instance, a procurement activity might involve both a 'purchase order' object and a 'vendor contract' object, culminating in a multi-object node that encapsulates this combined interaction.
- *Condition Node*: Captures specific conditions or constraints that might affect the progression of an activity or decision-making points.
- *Source Node*: Denotes the origin or provenance of data for a particular object or activity, ensuring rigorous traceability. For example, a 'Source' node might link to an 'Object' node to provide clarity on whether the data was sourced from an internal system, third-party vendor, or manual input.

2. **Inclusion of additional relations**: This involves introducing new relational connections between nodes to better represent interactions or dependencies. Examples of such relations include:
   - *Causal Relation*: Highlights the cause-and-effect relationships between activities or objects. For instance, if one activity's completion invariably triggers another, a causal relation can illustrate this dependency.
   - *SupervisedBy Relation*: Establishes a hierarchical connection between actors, indicating oversight or management. For instance, an 'Actor' might be 'supervised by' another 'Actor', denoting responsibility or oversight.
   - *InteractedWith*: A nuanced relation emphasizing the engagement level of an actor with an object. For instance, if an actor has viewed, modified, or referenced an object, this relation captures that interaction.
3. **Attributes Enrichment**: This involves enhancing nodes with additional metadata or properties to provide deeper insights into the characteristics of each entity. Examples of properties that could be added to nodes include:
   - *For the Activity Node*: Attributes like 'duration' can specify the time taken for completion, while 'cost' might denote resources or financial expenditure associated with that activity.
   - *For the Object Node*: Incorporating a 'status' attribute can provide real-time or historical insights into the object's lifecycle or progression.
   - *For Relations*: Enriching relationships with attributes can amplify their informational depth. For example, The 'FollowingBy' relation between two activities might have a 'waiting time' attribute, showcasing any lag or downtime between the end of one activity and the start of the next.

Incorporating these instance-level refinements holds the promise of elevating the metamodel's capabilities, rendering it a precise tool for navigating the extensive realm of process execution data. It's important to note that this exploration is not exhaustive; the landscape of enhancements is vast, offering numerous avenues for further refinement based on specific analytical requirements.

6.2.2 Process-level extensions

Merging instance and process-level insights into one comprehensive graph meta-model provides a panoramic view of process execution, ensuring no detail is overlooked. By seamlessly integrating these layers, the enhanced meta-model not only encapsulates the intricate specifics of individual events but also the broader, aggregated narratives that arise from clustering such data. For example, while individual nodes may detail specific process events, composite nodes could aggregate common sequences of activities or prevalent patterns. Similarly, relations can expand to highlight both individual instance pathways and overarching process trends. This enriched integration translates to a meta-model adept at offering both granular and macroscopic insights, marrying the specificity of individual instances with the overarching narratives of process-level data.

Incorporating aggregation aspects into the meta-model is a flexible approach, depending on the specific analytical needs. For instance, consider the graph meta-model introduced in [55]. In this model, the authors introduce a node type called "Class" with a "type" attribute. All events sharing the same attribute value are linked to the same "Class" node. Additionally, the authors propose "directly follow" relations between "Class" nodes to represent aggregated "directly follows" relations between events. This schema illustrates how aggregation can be integrated into the meta-model to facilitate different aspects of analysis.

6.3 Enhancements in Natural Language Querying of Process Execution Data

In our thesis, the second significant contribution pertains to allowing users to query process execution data through natural language, subsequently translating their queries into the Cypher language for our graph meta-model. While the current framework is a robust solution for this purpose, its potential is far from fully tapped. This section delves deep into the possible extensions and refinements for our natural language querying mechanism. It presents some possible perspective to use our work in a more advanced context. We center our focus on four key perspectives:

- **Expanding Query Versatility:** To enhance the versatility and user-friendliness of our system, broadening the spectrum of supported NL queries takes center stage. This expansion not only amplifies the utility for experienced users but also makes it more accessible for newcomers. Therefore, in Section 6.3.1, we meticulously examine the range of supported NL queries and investigate necessary interface adjustments to accommodate additional types of queries.
- **Adaptability to Multiple Graph Meta-Models:** While our current NLI framework is tailored for our proposed graph meta-model, the dynamic landscape of data necessitates adaptability. The ability to cater to multiple graph meta-models can significantly enhance the relevance and applicability of our system. Consequently, Section 6.3.2

delves into potential modifications and adaptations that could seamlessly accommodate different graph meta-models with minimal adjustments required.

- **Leveraging Large Language Models:** LLMs with prompt engineering offers an exciting avenue to further automate and optimize the translation of NL queries to Cypher. Harnessing the power of LLM can not only streamline the query translation process but also potentially improve its accuracy. In Section 6.3.3, we explore this promising approach, discussing its integration, benefits, and potential pitfalls.
- **Transitioning to a Conversational Interface:** Although our present approach delves into an NLI where NL queries yield responses, the prospect of incorporating conversational context, engaging with user dialogue history, and implementing responsive mechanisms is intriguing. Section 6.3.4 delves into the exploration and discussion of evolving our existing NLI into a conversational interface. This shift aligns with the dynamic trends in human-computer interaction, promising more interactive and intuitive user experiences.

These three perspectives collectively contribute to a deeper understanding of how our natural language querying system can evolve and adapt to the evolving landscape of data interaction, making it more versatile, adaptable, and efficient for a wide range of users and scenarios.

6.3.1 Broadening Supported NL Queries

In our pursuit to develop a comprehensive NLI system, we initially classified NL queries related to process execution data into three main categories: content, behavioral, and performance, as elaborated in Section 4.5. Currently, our NLI system efficiently caters to content and behavioral queries. Yet, there lies a spectrum of unexplored categories that could emerge based on specific analysis contexts. For instance, other potential category could be comparative queries which are designed to compare different attributes, periods, or instances of processes (e.g. *"How did process X perform in January compared to February?"*). To maximize the utility and adaptability of our interface, it's paramount to broaden the spectrum of supported NL queries. Doing so not only amplifies its immediate applicability but also positions our interface as a foundational tool for researchers. Scholars can utilize our system as a starting point, and expand upon it, potentially paving the way for the support of intricate and advanced NL queries. In light of this vision, this section outlines a systematic process that aids users and researchers in enhancing the interface with an extended range of supported NL queries.

The proposed expansion process is depicted in Figure. 6.2. It consists of four primary steps:

- **Query Categories Definition:** Initially, it is crucial to define the new categories of NL queries that the system should be equipped to handle. These categories would define the scope and nature of the interactions users will have with the system.

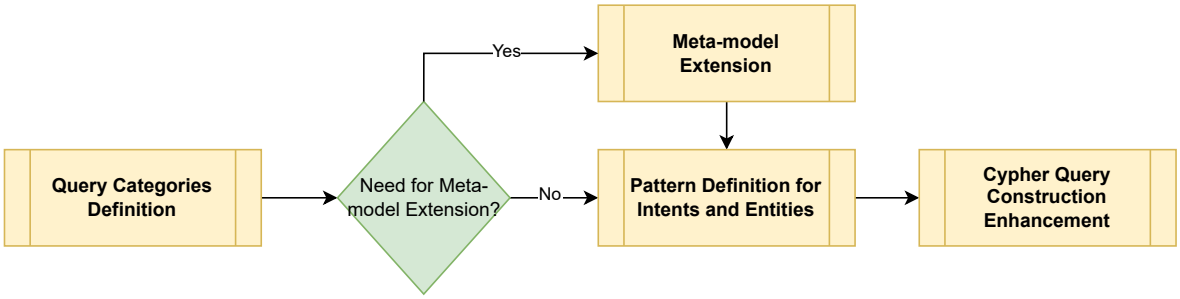


Figure 6.2: Four-Step Expansion Process for Enhancing Supported NL queries in our NLI

- **Meta-model Extension (if required):** Depending on the complexity and nature of the newly introduced NL query categories, there might be a need to extend the graph meta-model. This augmentation involves integrating additional elements essential for furnishing accurate and relevant answers to the proposed NL queries.
- **Pattern Definition for Intents and Entities:** For every identified NL query category, it's imperative to define a general pattern concerning its intent and associated entities. This approach mirrors our earlier methodology for content and behavioral queries. By understanding the underlying pattern, the system can more effectively discern and respond to user queries.
- **Cypher Query Construction Enhancement:** Once the intents and entities are demarcated, the subsequent step involves updating the query construction component. By introducing necessary rules, this component can adeptly construct Cypher query patterns, drawing from the predefined intents and entities.

By adhering to this structured process, one can seamlessly integrate a plethora of new NL queries, ensuring that our system remains adaptable, relevant, and at the forefront of NLI research.

6.3.2 Improving applicability to multiple graph metamodels

In the dynamic landscape of data modeling, users could employ a diverse array of meta-models based LPG to curate process execution data. These meta-models, while operating under the foundational principles of LPG, might differ in aspects like node types, relationships, properties, and other structural nuances. Therefore, it is of paramount importance that our NLI system retains the flexibility to accommodate these varied meta-models, ensuring broader relevance and applicability. As previously elaborated in Chapter 4, our NLU component has been meticulously designed with a degree of generalization. By proposing universally applicable intents and entity patterns that align with the LPG model, this component possesses the innate capability to seamlessly adapt to new graph meta-models. However, a significant challenge arises with the query construction component. Governed by a rule-based approach, this component has an inherent dependency on the meta-model's specific structure to accurately craft the corresponding Cypher queries.

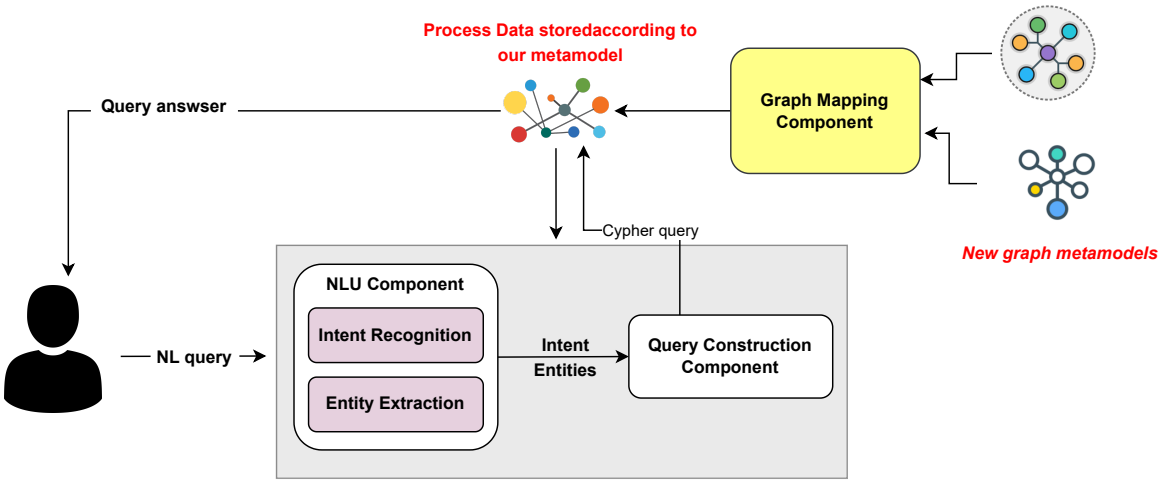


Figure 6.3: Mapping layer approach to strengthen the adaptability of our NLI system for new graph metamodels

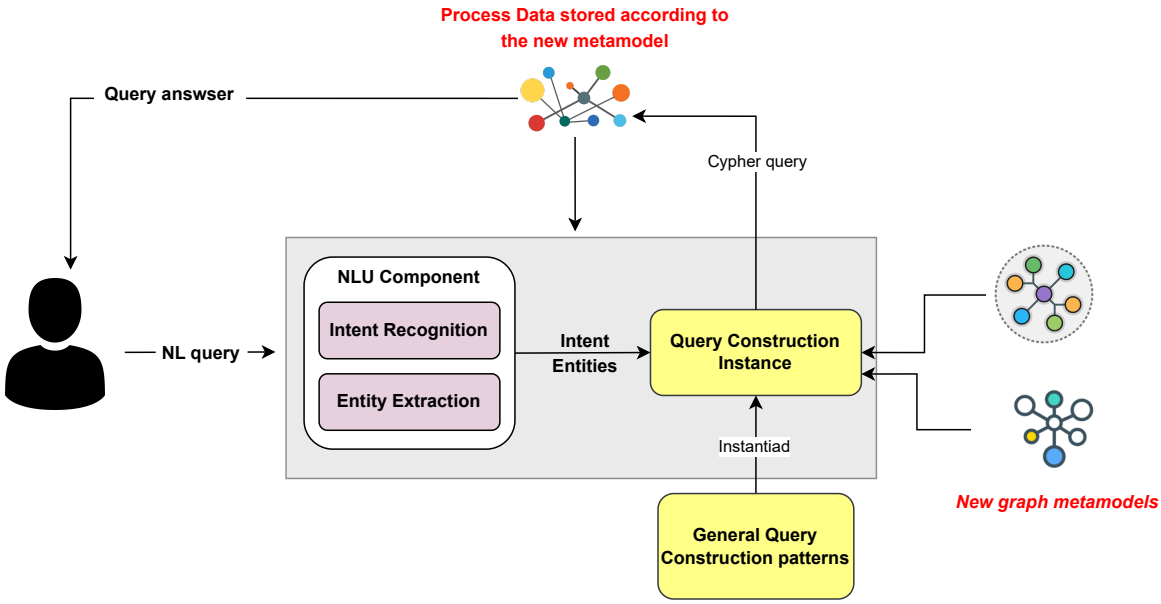


Figure 6.4: Generalized query construction approach to strengthen the adaptability of our NLI system for new graph metamodels

In pursuit of bolstering the system's adaptability, this section embarks on an exploration of two potential strategies to modify our NLI, thereby facilitating its integration with an extensive array of graph meta-models.

- **Mapping Layer Approach:** The first solution remains largely non-intrusive, preserving the integrity of the existing query construction component. Here, we retain the proposed NLI in its original form but introduce an intermediary mapping layer as depicted in Figure. 6.3. This layer serves as a translation mechanism, converting elements from the novel graph meta-model into elements synonymous with our defined meta-model. This approach is especially viable when direct mapping between the new and our meta-model does not culminate in any loss of context or information.

- **Generalized Query Construction:** The alternative proposition seeks a more comprehensive revamp, necessitating the formulation of a novel query construction component. This approach is illustrated in Figure. 6.4. Rooted in universally recognized patterns of the LPG, this component is designed to be inherently adaptable. Upon the introduction of a new graph meta-model, an additional instantiation layer is integrated. This layer, informed by the meta-model's specific structure, guides the instantiation of the query construction component, ensuring its alignment with the introduced meta-model.

While these solutions offer promising avenues, it's crucial to acknowledge that they represent the initial foray into an expansive domain. Alongside the opportunities they present, they also introduce a new set of challenges that need to be addressed. Future research endeavors should delve deeper, rigorously evaluating, comparing, and refining these propositions. Such investigative pursuits not only validate our solutions but also provide a foundational bedrock for subsequent research, inspiring and guiding scholars as they navigate this intricate realm of adaptability in graph meta-models and grapple with the emergent challenges.

6.3.3 Leveraging LLM with prompt engineering for constructing Cypher queries

Recent strides in the field of LLMs have opened up new horizons for re-envisioning our proposed solution. The sheer power and versatility of LLMs, combined with nuanced prompt engineering, offer a compelling alternative to traditional query construction methodologies. Such promising approaches hark back to previous chapters where we explored the potency of LLMs for the automated generation NL queries (Section 4.8), and Cypher queries tailored to process mining services (Section 5.6). By judiciously crafting high-quality prompts, we aim to lean on LLMs for the automatic conversion of NL inquiries into Cypher queries. The vision here is straightforward: pose a question in natural language to the LLM, let it construct the appropriate Cypher query, execute said query on the graph, and relay the result.

While the potential benefits of such an approach are confusing, they do not come without their fair share of challenges:

- **Domain-Specific Variability:** Process data, by its very nature, is entrenched in the specifics of its domain. Processes and their associated data can vary wildly from one context to another. Consequently, a one-size-fits-all approach is untenable. LLMs need tailored guidance—via the graph meta-model—to churn out the right Cypher query. In essence, a unique prompt might be necessary for each distinct process domain to ensure the LLM constructs an appropriate query.
- **Contextual Considerations:** Beyond domain-specific quirks, there is also the overarching matter of context. NL queries are often laden with nuances, and deciphering these nuances accurately is pivotal. For instance, assigning the right value to the correct property is not just about understanding the language, it is about grasping the context. An LLM would need to be equipped with an ample contextual background to discern and allocate the right values to their corresponding properties.

In conclusion, leveraging LLMs with prompt engineering to construct Cypher queries offers a compelling research avenue, it is by no means a panacea. It comes bundled with challenges that warrant thorough scrutiny, investigation, and refinement. Addressing these challenges is imperative to unlock the true potential of this approach and to ensure its feasibility and accuracy in real-world applications. Looking forward, one potential perspective lies in the exploration of customized LLMs tailored specifically to our approach. While existing works primarily focus on utilizing pre-existing LLMs with prompt engineering for designated tasks, the emergence of future LLM generations customized for specific applications raises intriguing possibilities. The development of such specialized LLMs could revolutionize the landscape, offering tailored solutions to our specific context and potentially overcoming some of the challenges associated with the current approach.

6.3.4 Transitioning to a Conversational Interface

In our current work, we acknowledge that the focus is on exploring an NLI, wherein a user provides an NL query and receives a response. Looking ahead, our future vision extends beyond this singular interaction model. We aspire to evolve our existing approach into a conversational interface, where the system not only responds to individual queries but also contextualizes these interactions within a broader conversation. This evolution entails considering the conversational history and implementing mechanisms to retain memory across interactions. By incorporating this conversational aspect, our system can engage in more dynamic, continuous, and meaningful dialogues with users, enhancing the user experience and expanding the capabilities of NL querying in the realm of process execution data. This shift toward a conversational interface aligns with the evolving trends in human-computer interaction, promising a more intuitive, interactive, and personalized user experience.

In envisioning this conversational evolution, we recognize that transitioning to such a sophisticated solution requires dedicated research and innovation in the fields of chatbots and conversational interfaces. Researchers and developers must explore advanced techniques in machine learning, dialogue modeling, and context-aware computing to create a seamless

and intuitive conversational experience. By investing in these specialized areas, we can pave the way for a transformative shift in how users interact with systems, ensuring that our conversational interface not only understands individual queries but also comprehends the nuances of continuous conversations, making the interaction more natural, engaging, and effective.

6.4 Advancements in Service-Oriented Architecture for Process Mining Techniques

In chapter 5 we introduced the architecture dedicated to the discovery and access of process mining techniques. Recognizing its criticality, our research has sought not just to design this architecture, but to also ensure it remains resilient, flexible, and adaptable to changes. Such forward-thinking design approaches are pivotal, as they can accommodate the dynamic nature of the process mining domain. This section delves into potential extensions and refinements that can be made to our proposed service-oriented architecture for process mining techniques. The goal is to elucidate the areas where the architecture can be enhanced for greater applicability and robustness. We divide this discussion into three primary subsections, addressing the service description, the REST API design, and the possibility of a transition to a more resource-oriented design.

6.4.1 Service description level enhancements

The vitality of any service-oriented architecture lies in its heart—the service description. It covers the range and depth of services offered. Ensuring that this central component remains comprehensive and continually relevant is paramount. One way to ensure this is by fostering an environment of constant introspection and adaptability. With the evolving nature of process mining, there is an ever-present need to reevaluate and expand our service description to capture the nuances of the domain.

In light of the aforementioned, there are two primary avenues for extension:

- **Process Mining Methods Extension:** The present graph metamodel efficiently represents discovery, conformance, and prediction methods. However, to maintain a holistic and up-to-date representation, one must consider expanding this scope. For instance, an additional type that can be integrated is the "process enhancement" method. To realize this, we can: (i) dive deep into established surveys in the field, gleaning insights that help in extracting properties unique to this domain; (ii) define a new node type within our process mining method property graph to encapsulate the distinctive characteristics of the method.
- **Properties Extension:** The existing metamodel incorporates general properties that are universally applicable across all methods. However, this universal approach, while

encompassing, may sometimes overlook the intricate details specific to sub-categories within each method. The idea here is to delve deeper, creating layers of categorizations and associating them with distinct properties. For illustration, consider the "discovery" methods. These can be further bifurcated into:

- Domain Knowledge-based Discovery: Methods falling under this are driven by domain-specific knowledge. They can be further characterized based on:
  - Nature of Domain Knowledge: Is it explicit, where rules and conventions are overtly stated, or is it implicit, relying on more subtle cues and insights?
  - User Interaction Level: Does it necessitate constant user input, or does it function with minimal user intervention?
- Non-Domain Knowledge-based Discovery: Methods here operate without heavy reliance on domain knowledge, offering a different approach and set of characteristics.

Such detailed categorizations and properties pave the way for a nuanced, multi-dimensional representation, fostering a deeper understanding and facilitating a more pinpointed service matching.

Based on the evolving needs of analysis and the dynamic nature of the process mining domain, it's evident that the existing process mining method property graph can be enriched. By integrating additional methods, diving deeper into categorizations, and emphasizing distinctive properties, we can significantly elevate the information representation, making the architecture more robust and adaptable.

6.4.2 Transitioning to resource-oriented REST API design

The world of REST API design offers multiple paradigms, each with its unique advantages and implications. Our present methodology, rooted in the service-oriented REST API design, predominantly underscores the functional aspects. This has offered clear interfaces that map directly to distinct functionalities, making them actionable and easily understandable. However, as with any technological domain, evolution is inevitable, and there is potential to explore alternatives that could bring forth additional benefits.

The resource-oriented design paradigm emerges as a promising alternative, shifting the focus from functional capabilities to the management of resources. In the context of process mining, these resources can encompass a wide range of outputs generated by various process mining methods. This approach often aligns more naturally with how users perceive and interact with data, resulting in a more intuitive API design. Additionally, it brings scalability benefits, enabling the seamless addition, modification, or removal of resources without causing ripple effects across the system, thereby preserving flexibility and scalability. Furthermore, standardizing operations around common HTTP methods such as GET, POST, PUT, and DELETE, it simplifies the consumption of the API.

Transitioning from our existing service-oriented design to a resource-oriented one involves several key steps:

- **Resource Identification:** The first step involves identifying all possible resources. In our context, these would be the diverse outputs generated by process mining methods.
- **Define Resource Relationships:** Understand how different resources relate to one another. This might involve defining hierarchies or mapping out associations.
- **URL Schema Redefinition:** Once resources are identified, URLs should be structured to reflect these resources rather than functions or services.
- **Update CRUD Operations:** Shift the focus from custom methods to more standardized CRUD (Create, Read, Update, Delete) operations, aligning with the typical HTTP methods.
- **Documentation & Training:**Revise API documentation to align with the resource-oriented structure and offer training or guidance to users and developers.

Transitioning from service-oriented to resource-oriented design is not merely a technical endeavor but also a conceptual one, fraught with ambiguities and challenges. Central to these challenges is the very definition of a 'resource'. What constitutes a resource can be seen differently based on the use case, organizational goals, or even user perspectives. The choice between service-oriented and resource-oriented designs is not merely technical; it's strategic. Decisions made in haste, without comprehensive consultations with users, developers, and organizational stakeholders, could lead to designs that, while technically sound, may not align with the actual needs and workflows of the users.

6.5 Conclusion

This chapter provides a meticulous examination of potential refinements and extensions to the core contributions of our thesis. The graph metamodel, currently serving as a dynamic framework for process data, is recognized not just for its present capabilities but more importantly for its potential adaptability. By envisioning broader integrations, the chapter advocates for the model's growth to incorporate more detailed process information. In parallel, the scope of our NLI system is assessed, emphasizing the importance of diversifying its range and enhancing its compatibility, especially through the potential integration of advanced tools like LLMs. Furthermore, the service-oriented architecture's potential enhancements are discussed, concentrating on its need for greater flexibility and suggesting a transition towards a more resource-oriented design approach. In essence, while the thesis offers a robust foundation, this chapter paints a future picture, where each component is further refined, expanded, and adapted to meet the ever-evolving demands of process analysis.

Chapter 7

Conclusion

In the ever-evolving landscape of organizational operations, business processes stand as the backbone, orchestrating a myriad of activities aimed at achieving organizational goals. These processes, spanning across various sectors, generate a treasure trove of data, encapsulated in event logs. Such data, rich in detail and scope, offers a unique window into the intricacies of business operations. Recognizing the goldmine of insights embedded within these logs, organizations have turned to process analysis. This discipline, rooted in data-driven decision-making, empowers organizations to refine their operations, identify inefficiencies, and foster a culture of continuous improvement.

Central to the realm of process analysis is the concept of process querying. This technique allows for the interactive retrieval and manipulation of process-related data. With the ability to delve into repositories filled with diverse process models, process querying offers a robust mechanism for managing and analyzing process data. However, the true power of process querying is realized when applied to process execution data, stored and represented using specialized techniques. However, the intricacies of query languages often act as barriers, especially for business professionals lacking technical expertise. The challenge extends beyond just understanding these languages; it also encompasses the selection of appropriate storage mechanisms for efficient querying. Process mining, another cornerstone of process analysis, offers a systematic approach to dissecting event logs to uncover real-world process behaviors. By leveraging algorithms and methods, organizations can unearth insights, detect inefficiencies, and identify areas of non-compliance. However, the practical application of process mining is not without its challenges. From selecting the right methods to ensuring compatibility with existing software tools, analysts often find themselves navigating a maze of complexities.

Incorporating cognitive capabilities into process analysis marks a significant stride toward making advanced analytical techniques more accessible and user-friendly. This thesis pivots towards AI-driven solutions, specifically a conversational AI framework for cognitive process analysis. By bridging the gap between complex analytical techniques and user accessibility, our research paves the way for a more intuitive, efficient, and effective process analysis landscape. The primary goal is to empower organizations with intuitive and user-friendly tools, allowing them to effortlessly scrutinize their business processes and extract valuable insights from process data. Our work revolves around two primary objectives aimed at facilitating cognitive process analysis for users:

- **Simplifying access to stored process data:** The first objective revolves around the simplification of accessing and querying process data. This encompasses the creation of user-centric interfaces and tools that not only render process data accessible but also make it intuitive for a wide range of users. This is achieved by providing an interface to automatically construct database queries from NL queries to access the process data.
- **Simplifying access process mining methods:** The second objective is centered on the facilitation of searching and applying process mining methods. Our aim is to establish a unified and user-friendly environment where process mining techniques are readily discoverable and accessible, irrespective of the technical intricacies involved or the users' level of proficiency.

Within the scope of the first objective, we have made two significant contributions:

- **Graph Metamodel for Effective Data Storage:** Our first contribution involves the proposal of a graph metamodel based on LPG to represent process data comprehensively. This metamodel incorporates various node types, relation types, and properties, facilitating the storage of diverse concepts and relationships related to multi-dimensional process data.
- **NL-Based Querying Pipeline:** To make process data naturally accessible to all users, we introduce a natural language-based pipeline that assists end users in querying process data stored according to our LPG model. This pipeline takes a user query, formulated in natural language, and automatically constructs a corresponding Cypher query for execution over structured process data. It then returns the response. This pipeline is a hybrid approach that combines machine learning and rule-based methods, incorporating two main stages: intent detection and entity extraction through machine learning in the first stage and a rule-based approach to generate the corresponding database query in the second stage. Furthermore, to ensure adaptability across various process domains with minimal user intervention, we proposed first general patterns for intents and entities, that are defined based on the labeled property graph meta-model in general and that are instantiated according to the event property graph model. Second, we proposed an automated approach to generate NL queries to train the NLU component. We leverage GPT-4's powerful language generation capabilities to automatically generate a wide range of NL queries without the need for a manually annotated dataset for each new domain.

For the second objective, we proposed a service-based solution to tackle the challenges faced by analysts when discovering and accessing process mining methods. The proposed solution leverages REST APIs to provide process mining services and is made up of three main components. The first component is services description, which employs a graph metamodel based on LPG to describe the available process mining methods. It includes the necessary concepts to represent services, such as the properties of each service, along with the required inputs and outputs. The second component is a unified service-oriented REST API design. It leverages the inherent service-oriented nature of process mining methods and provides users

with a comprehensive and cohesive framework to access and utilize various functionalities. The final component is services matching, which matches the consumer's requirements expressed in NL with suitable process mining services.

For a comprehensive validation of the graph metamodel, we utilized two distinct datasets that are publicly accessible and stored in both CSV and OCEL formats. The same datasets were instrumental in assessing the efficacy of our NL querying pipeline. In this endeavor, we collected over 370 NL queries from users and, with the aid of paraphrasing tools, produced an additional 150 NL queries. Furthermore, we embarked on a proof-of-concept evaluation of our service-oriented architecture. We used more than 110 NL queries to evaluate the service matching component. These NL queries are specifically tailored to descriptions of process mining services. Additionally, we carried out a use case study with external participants to gauge user experience in searching and accessing process mining methods and to gather feedback.

The design principles we presented in the introduction (Section 1.3.1) have been respected:

- User-Centric Design (Principle 1): Our approach has been meticulously crafted with the end-user in mind. We have proposed solutions that make process analysis tasks accessible to all users with diverse levels of expertise. First, we proposed the natural language-based querying pipeline to make process data naturally accessible to users without the need for complex technical knowledge. Second, we proposed a service-oriented architecture that enhances the ease of discovering and accessing process mining services, allowing users to make requests using natural language.

- Automation (Principle 2): Automation stands at the core of our contributions. The natural language-based querying pipeline automates the translation of user queries into structured database queries, eliminating the need for manual translation. Furthermore, we integrated an automated solution for generating NL training data used in this pipeline. Additionally, our service-based solution for process mining, automates the discovery and access to various process mining methods, ensuring swift and efficient operations.

- Adaptability & Generality (Principle 3): Our solutions are designed to be both adaptable and general. The graph metamodel based on LPG is versatile, accommodating diverse concepts and relationships related to multi-dimensional process data. The natural language-based querying pipeline, with its general patterns for intents and entities, can be instantiated across various process domains, ensuring broad applicability. Our service-based solution, with its versatile service description component and unified API design, is poised to adapt to evolving user needs and the ever-growing landscape of process mining methods.

Last but not least, the solutions presented in this thesis are not just static or rigid constructs; they are inherently extensible. This extensibility ensures that as the landscape of the process analysis domain evolves, the solutions can evolve with it. Their flexibility ensures

that they can be molded or adjusted to cater to a range of different scenarios or requirements without a complete overhaul. Furthermore, adaptability is a key feature of our solutions. As new techniques emerge, or as the demands of users change, the solutions can be modified to incorporate these new elements. For instance, the graph metamodel designed for storing process data can be enhanced with complementary process-centric details, with the aim of enhancing its comprehensiveness and broadening its scope of analytical aspects. Similarly, the service description component holds potential for expansion, allowing for the integration of an increased array of process mining techniques and attributes. By designing solutions that can adapt and grow, we ensure longevity and continued relevance. This adaptability not only ensures the utility of the solutions in the present but also secures their applicability in the future, making them invaluable assets in the dynamic world of process mining.

Appendix A

Prompt Engineering

A.1 Prompt for automated generation of NL queries

In our work we developed a pipeline that takes a natural language user query as input, translates it into a Cypher query, and executes it on an event property graph to obtain the query result, which is then returned to the user. The pipeline comprises two main components: the NLU component and the query construction component.

NLU component: This component receives the user's query and performs Natural Language Understanding (NLU) tasks. It allows to detect the user's intent (i.e. what the user is asking about) and to extract named entities from the query (i.e. terms that correspond to elements in the event property graph). In this work, we adopt a machine learning-based approach for intent recognition and entity extraction since it is more robust against variations in natural language.

In machine learning, intent recognition and entity extraction can be seen as a classification task in which a model is trained with a set of queries and their associated intents and entities. Therefore, the main contribution of this component is the identification of the possible intents and entities that the system should learn to detect. In our work, as we aim to query process data, intents and entities should pertain to elements in our event property graph. Intents describe the type of information to be searched for and returned to the user, such as nodes, relationships, paths, node/relationship attributes, or any aggregation applied to numeric attributes. Entities refer to graph elements (e.g., node types, node attributes, relation types).

We propose general patterns that are defined based on the labeled property graph metamodel in general and that are instantiated according to the event property graph model. Once defined, we can apply state of the art classification techniques for intent recognition and entity extraction. However, one main challenge of this work is the generation of training data that trained the machine learning model with natural language queries associated with their intents and entities.

Now I will explain to you the general intent pattern and they are instantiated to each domain. To identify intent classes, we divide process data NL queries into three main categories based on state-of-the-art process querying and process mining techniques: content, behavioral, and performance. Each of these categories requires dedicated intent definition

and Cypher query construction mechanisms. The developed pipeline supports content and behavioral queries. Content NL queries allow the user to either ask about particular nodes without taking into account any of their relationships, or about particular nodes that are involved in specific relationships with other nodes. For instance, in the question [Which applications have a requested amount more than 15000], it searches for the [Application] nodes whose amount attribute is greater than 15000. In the question [Which accepted applications have a requested amount more than 15000], it searches for the [Application] nodes connected, through the relation [Affect], to the [Activity] nodes whose name is [A_accepted]. Therefore, we define two classes of content intent patterns: [Node] patterns and [Node_Relation] patterns. Both patterns indicate that the user is seeking information about the node or its attributes. Given the proposed event property graph, these patterns can be instantiated according to each graph model. For example, the intent pattern [Node] is instantiated to each node type in the event property graph, e.g. intents [Actor], [Activity], [Application] and [Offer]. Similarly, the intent [Node_Relation] is instantiated to every possible node and relationship type e.g. intents [Actor_Contribute], [Activity_Affect], etc. It is worth noting that for [Node_Relation] pattern instantiation, we add all possible pairs regardless of the relation direction. For instance, the intents [Actor_Contribute] and [Activity_Contribute] are two possible instantiations. The former is linked to queries about the actors executing a specific activity while the latter is linked to queries about the activities executed by specific actor.

Two additional intent patterns, [Agg_Node] and [Agg_Node_Relation], are defined on top of the [Node] and [Node_Relation] patterns. These two patterns are similar to the first two patterns, but instead of reflecting the user's want for information about a node or its properties, they indicate that the user inquires for the result of one of the aggregate functions: [maximum, minimum, average, count or sum]. For instance, the question [What is the maximum requested amount for car loan applications?], the user inquires about the result of the [max] function applied on the attribute [amount] of the [Application] nodes whose [Goal] is equal to ['car loan']. Its associated intent is [Max_Application] which is instantiated from the intent pattern [Agg_Node]. Now as I previousely stated the main challenge is the generation of natural language for training data. Thus one possible solution is to define general templates for each general intent pattern (i.e. node, node_relation, agg_node, agg_node_relation) and then instantiate these template by replacing placeholder with specific values, and at the end to agent the training data we used language models or paraphrasing tool. For, example considering the intent pattern : node, one template that we could defined for this pattern: Give me all [specific_intent] with [attribute_1]. I will give you a set of intent each with description and related attribute you are asked to generate generale templates for each intent and then from these templates gerenate more than 50 natural language queries.

Here is the first intent:
intent="application:" description: the user ask about application with specific condition
application_attribute: requested amount, application ID, loan goal
application_attribute_values: requested amount: numerical value, application ID: has the form Application_number, loan goal :[car, home improvement, investigation].

A.2 Prompt for matching services

Our research work proposes a component called service description" that utilizes a labeled property graph to describe process mining services, specifically discovery, conformance, and prediction. The goal is to query this model to find services that align with user needs. In order to identify these services, we need to define the distinguishing properties, inputs, and outputs for each type of service. To accomplish this, we introduce an LPG graph metamodel that characterizes information related to discovery, conformance, and prediction methods. This metamodel serves as a foundation for discovering services regardless of the technology used. The graph consists of six different entity types. Three of these entity types represent the three main techniques of process mining: Discovery, Conformance, and Prediction.

1. The Discovery entity represents a process discovery method and has the following properties:

- Name
- modelType: signifies the class of the discovered process model (could have the values: procedural, declarative, or hybrid)
- modelFormalism: specifies the specific notation or formalism of the discovered process model (could have values such as PetriNet, BPMN, declare, etc.)
- algorithm: indicates the algorithm used to discover the model (could have values such as alpha miner, inductive miner, etc.)
- isDomainKnowledge: shows whether the method requires domain knowledge, meaning if it needs extra information about the domain, alongside the event log, for the discovery of the process model (could have the values: yes or no)
- applicationDomain: indicates the array of application domains where the method has been applied and evaluated (could have values such as health care, business management, loan application, etc.)

2. The Conformance entity represents a conformance-checking method and has the following properties:

- Name
- modelingLanguage denotes the language employed to describe the process behavior, serving as input to the conformance-checking algorithm (could have values such as: PetriNet, BPMN, declare, etc.)
- perspective: specifies the minimum information that the event log file must contain for the conformance method to be executed (could have values such as: control flow, multi-perspective, etc.)
- algorithmType: specifies the algorithm employed to compare the process model to the event log (could have values such as: trace alignment, log replay, etc.)
- metrics: outlines the array of metrics that are outputted by the method to represent conformance (could have an array of values from Fitness, Precision, Simplicity, Generality)

3. The Prediction entity represents a prediction method and has the following properties:

- Name
- Task: defines the type of operation that the prediction method performs (could have the following values: build-model or predict)
- predictionType: refers to the aspects of the business process the method predicts (could have the following values: next activity, outcome, LTL, remaining time)
- awareness: indicates whether the method is process-aware or not (could have the following values: PA which refers to process-aware or NPA which refers to non-process aware)
- problemType (could have the following values: classification or regression)
- methodology: indicates the methodology used to build the predictor model (could have values such as: machine learning, annotated transition system (ATS), statistical (STAT), etc.)
- algorithm: indicates the array of algorithms employed (could have an array of values such as decision tree, clustering support vector machine, etc.)

4. The inputs and outputs of each method are represented by Input and Output entities, connected to the Discovery, Conformance, and Prediction entities through the relations hasInput and hasOutput, respectively. The Input entity has the properties:

- Name: specifies the name of the input (such as event log, unit time, configurable parameters, etc.)
- Type: specifies the type of input (could have the values: literal or resource)

5. Similarly, the Output entity has the name and format properties.

6. Additionally, since process mining methods can be complex and consist of multiple functions, a Module entity is used to represent the functional sub-modules involved in these methods. Each method entity is connected to the corresponding Module entity through a submodule relation. Each Module can also have its own inputs and outputs. It is important to note that the method for making predictions is typically based on methods for building the predictor model. To capture this relationship, we include a building-model relation in the metamodel.

The objective of this task is to find suitable services that meet the requirements of users by searching through stored services using the Cypher Language. We will receive user requests in natural language and convert them into Cypher queries. These queries should retrieve the services, along with their associated inputs, outputs, and submodules. The query should also optionally match the modules related to the services as well as their associated inputs and outputs and return them. For instance, if a user asks for discovery methods that generate a Petri Net model, the corresponding Cypher query should resemble a specific MATCH statement as follows:
MATCH (discovery: Discovery modelingFormalism: 'PetriNet')-[:hasInput]-(input:Input), (discovery)-[:hasOutput]-(output:Output)
OPTIONAL MATCH (discovery)-[:subModule]-(module:Module)

OPTIONAL MATCH (output1:Output)-[:hasOutput]-(module)-[:hasInput]-(input1:Input)
RETURN DISTINCT (discovery),(input),(output),(module),(input1),(output1).

Similarly, if a user requests prediction services related to remaining time using machine learning, another MATCH statement should be created:
MATCH (prediction: Prediction predictionType: "remaining time", methodology: "machine learning")-[:hasInput]-(input:Input), (prediction)-[:hasOutput]-(output:Output)
OPTIONAL MATCH (prediction)-[:subModule]-(module:Module)
OPTIONAL MATCH (output1:Output)-[:hasOutput]-(module)-[:hasInput]-(input1:Input)
RETURN DISTINCT (prediction),(input),(output),(module),(input1),(output1).

Even when users inquire about results, we must identify the services that produce those results and generate a Cypher query for them. The generated Cypher queries should be syntactically correct and search in addition to the services, their input, output, submodules and submodules inputs and outputs. We will provide a set of user requests, and for each request, you should generate the corresponding Cypher query. Please note that if a property contains an array of values, you should look for a specific value(s) within that array. While constructing these queries, please refrain from introducing new node labels or attributes. Know for each user request that we provide generate the corresponding Cypher query without explanation.

Appendix B

List of Publications

- **Journal articles**

  1. Meriana Kobeissi, Nour Assy, Walid Gaaloul, Bruno Defude, Boualem Benatalla, Bassem Haidar. "Natural language querying of process execution data". In: Information Systems 116 (2023), p. 102227 (cit. on p. 53).

- **Conference articles**

  1. Meriana Kobeissi, Nour Assy, Walid Gaaloul, Bruno Defude, Bassem Haidar. "An intent-based natural language interface for querying process execution data". In: 2021 3rd International Conference on Process Mining (ICPM). IEEE. 2021, pp. 152–159 (cit. on p. 53).
  2. Ali Nour Eldin, Nour Assy, Meriana Kobeissi, Jonathan Baudot, Walid Gaaloul: Enabling Multi-process Discovery on Graph Databases. CoopIS 2022: 112-130

Appendix C

Proof of Concepts

---

1. "Natural language querying of process execution data", at https://www-inf.telecom-sudparis.eu/SIMBAD/tools/ProcessNLI/
2. "Service-oriented architecture for discovering and accessing process mining techniques", an initial version of this architecture focusing on prediction techniques at https://github.com/merianakb/SOA.git
3. Implemented Rest APIs for three prediction methods available in ProM at https://anonymous.4open.science/r/Web-Service-Oriented-Architecture-for-Discovering-and-Accessing-Predictive-Methods–2D54/README.md

Bibliography

[1] Wil M. P. van der Aalst. "Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data". In: *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*. Ed. by Peter Csaba Ölveczky and Gwen Salaün. Vol. 11724. Lecture Notes in Computer Science. Springer, 2019, pp. 3–25 (cit. on p. 18).

[2] Asma Adala, Nabil Tabbane, and Sami Tabbane. "A framework for automatic web service discovery based on semantics and NLP techniques". In: *Advances in Multimedia* 2011 (2011), pp. 1–1 (cit. on p. 34).

[3] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. "A comparative survey of recent natural language interfaces for databases". In: *VLDB J.* 28.5 (2019), pp. 793–819 (cit. on pp. 21, 22).

[4] Rama Akkiraju et al. "Web service semantics-wsdl-s". In: (2005) (cit. on p. 33).

[5] Rosa Alarcon et al. "REST web service description for graph-based service discovery". In: *Engineering the Web in the Big Data Era: 15th International Conference, ICWE 2015, Rotterdam, The Netherlands, June 23-26, 2015, Proceedings 15*. Springer. 2015, pp. 461–478 (cit. on pp. 31, 33).

[6] Gustavo Alonso et al. *Web services*. Springer, 2004 (cit. on p. 31).

[7] Renzo Angles. "The Property Graph Database Model". In: *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management*. Vol. 2100. CEUR Workshop Proceedings. 2018 (cit. on pp. 13, 19, 44, 88).

[8] Idir Aoudia et al. "Service composition approaches for internet of things: a review". In: *International Journal of Communication Networks and Distributed Systems* 23.2 (2019), pp. 194–230 (cit. on p. 32).

[9] Adriano Augusto et al. "Automated discovery of process models from event logs: Review and benchmark". In: *IEEE transactions on knowledge and data engineering* 31.4 (2018), pp. 686–705 (cit. on pp. 10, 28, 30, 65, 88, 97).

[10] Ahmed Awad. "BPMN-Q: A language to query business processes". In: (2007) (cit. on p. 4).

[11] Ahmed Awad, Gero Decker, and Mathias Weske. "Efficient Compliance Checking Using BPMN-Q and Temporal Logic". In: *Business Process Management, 6th International Conference, BPM*. Vol. 5240. 2008, pp. 326–341 (cit. on p. 18).

[12] Thais A Baldissera and Luis M Camarinha-Matos. "SCoPE: service composition and personalization environment". In: *Applied Sciences* 8.11 (2018), p. 2297 (cit. on p. 32).

[13] Luciana Barbieri et al. "A natural language querying interface for process mining". In: *Journal of Intelligent Information Systems* (2022), pp. 1–30 (cit. on pp. 25, 26, 28).

[14] Luciana Barbieri et al. “Towards a Natural Language Conversational Interface for Process Mining”. In: *International Conference on Process Mining*. Springer, Cham. 2022, pp. 268–280 (cit. on pp. 25, 26, 28, 84).

[15] Moshe Chai Barukh et al. “Cognitive augmentation in processes”. In: *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future: Essays Dedicated to Michael Papazoglou on the Occasion of His 65th Birthday and His Retirement* (2021), pp. 123–137 (cit. on p. 5).

[16] Catriel Beeri et al. “Monitoring Business Processes with Queries”. In: *Proceedings of the 33rd International Conference on Very Large Data Bases*. 2007, pp. 603–614 (cit. on p. 18).

[17] Catriel Beeri et al. “Querying business processes with BP-QL”. In: *Inf. Syst.* 33.6 (2008), pp. 477–507 (cit. on pp. 4, 18).

[18] Amin Beheshti, Boualem Benatallah, and Hamid Reza Motahari-Nezhad. “Processatlas: A scalable and extensible platform for business process analytics”. In: *Software: Practice and Experience* 48.4 (2018), pp. 842–866 (cit. on p. 19).

[19] Seyed-Mehdi-Reza Beheshti, Boualem Benatallah, and Hamid Reza Motahari-Nezhad. “Scalable graph-based OLAP analytics over process execution data”. In: *Distributed and Parallel Databases* 34 (2016), pp. 379–423 (cit. on p. 19).

[20] Seyed-Mehdi-Reza Beheshti et al. “A Query Language for Analyzing Business Processes Execution”. In: *Business Process Management - 9th International Conference*. Vol. 6896. 2011, pp. 281–297 (cit. on pp. 4, 9, 18, 19).

[21] Patrizio Bellan, Mauro Dragoni, and Chiara Ghidini. “Process extraction from text: state of the art and challenges for the future”. In: *arXiv preprint arXiv:2110.03754* (2021) (cit. on p. 25).

[22] Francesca Benzi, Dario Maio, and Stefano Rizzi. “VISIONARY: a Viewpoint-based Visual Language for Querying Relational Databases”. In: *J. Vis. Lang. Comput.* 10.2 (1999), pp. 117–145 (cit. on p. 21).

[23] Sonia Bergamaschi et al. “Combining user and database perspective for solving keyword queries over relational databases”. In: *Inf. Syst.* 55 (2016), pp. 1–19 (cit. on p. 22).

[24] Alessandro Berti and Wil van Der Aalst. “Extracting multiple viewpoint models from relational databases”. In: *International Symposium on Data-Driven Process Discovery and Analysis*. Springer. 2018, pp. 24–51 (cit. on p. 19).

[25] Alessandro Berti and Mahnaz Sadat Qafari. “Leveraging Large Language Models (LLMs) for Process Mining (Technical Report)”. In: *arXiv preprint arXiv:2307.12701* (2023) (cit. on pp. 27, 28).

[26] Alessandro Berti, Daniel Schuster, and Wil MP van der Aalst. “Abstractions, Scenarios, and Prompt Definitions for Process Mining with LLMs: A Case Study”. In: *arXiv preprint arXiv:2307.02194* (2023) (cit. on pp. 27, 28).

[27] Alessandro Berti, Sebastiaan J Van Zelst, and Wil van der Aalst. "Process mining for python (PM4Py): bridging the gap between process-and data science". In: *arXiv preprint arXiv:1905.06169* (2019) (cit. on pp. 5, 29, 30, 88).

[28] Alessandro Berti, Sebastiaan J van Zelst, and Wil MP van der Aalst. "PM4Py Web Services: Easy Development, Integration and Deployment of Process Mining Features in any Application Stack." In: *BPM (PhD/Demos)*. 2019 (cit. on pp. 29, 30, 88).

[29] Lukas Blunschi et al. "SODA: Generating SQL for Business Users". In: *Proc. VLDB Endow.* 5.10 (2012), pp. 932–943 (cit. on pp. 21, 22, 24, 71, 72).

[30] Sara Bouguelia et al. "Context knowledge-aware recognition of composite intents in task-oriented human-bot conversations". In: *International Conference on Advanced Information Systems Engineering*. Springer. 2022, pp. 237–252 (cit. on p. 23).

[31] David J Brenes, Daniel Gayo-Avello, and Kilian Pérez-González. "Survey and evaluation of query intent detection methods". In: *Proceedings of the 2009 Workshop on Web Search Click Data*. 2009, pp. 1–7 (cit. on p. 22).

[32] Dominic Breuker et al. "Comprehensible predictive models for business processes". In: *Mis Quarterly* 40.4 (2016), pp. 1009–1034 (cit. on p. 115).

[33] Dominic Breuker et al. "Designing and evaluating an interpretable predictive modeling technique for business processes". In: *Business Process Management Workshops: BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers 12*. Springer. 2015, pp. 541–553 (cit. on p. 115).

[34] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901 (cit. on p. 27).

[35] Tiziana Catarci, Mariano Leva, and Massimo Mecella. "Visual Query Languages for Data Stream". In: *Encyclopedia of Database Systems, Second Edition*. Ed. by Ling Liu and M. Tamer Özsu. Springer, 2018 (cit. on p. 21).

[36] Ryan H. Choi and Raymond K. Wong. "VXQ: A visual query language for XML data". In: *Inf. Syst. Frontiers* 17.4 (2015), pp. 961–981 (cit. on p. 21).

[37] Erik Christensen et al. *Web services description language (WSDL) 1.1*. 2001 (cit. on p. 31).

[38] Benjamin Clavié et al. "Large Language Models in the Workplace: A Case Study on Prompt Engineering for Job Type Classification". In: *International Conference on Applications of Natural Language to Information Systems*. Springer. 2023, pp. 3–17 (cit. on p. 27).

[39] Raffaele Conforti et al. "PRISM–a predictive risk monitoring approach for business processes". In: *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings 14*. Springer. 2016, pp. 383–400 (cit. on p. 115).

[40] Massimiliano De Leoni, Wil MP Van der Aalst, and Marcus Dees. "A general framework for correlating business process characteristics". In: *International Conference on Business Process Management*. Springer. 2014, pp. 250–266 (cit. on pp. 109, 114, 115).

[41] Patrick Delfmann et al. “The generic model query language GMQL - Conceptual specification, implementation, and runtime evaluation”. In: *Inf. Syst.* 47 (2015), pp. 129–177 (cit. on p. 18).

[42] Daniel Deutch and Tova Milo. “On models and query languages for probabilistic processes”. In: *SIGMOD Rec.* 39.2 (2010), pp. 27–38 (cit. on p. 18).

[43] Daniel Deutch and Tova Milo. “Type inference and type checking for queries on execution traces”. In: *Proc. VLDB Endow.* 1.1 (2008), pp. 352–363 (cit. on p. 18).

[44] Daniel Deutch and Tova Milo. “Type inference and type checking for queries on execution traces”. In: *Proc. VLDB Endow.* 1.1 (2008), pp. 352–363 (cit. on p. 18).

[45] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT (1)*. Association for Computational Linguistics, 2019, pp. 4171–4186 (cit. on p. 23).

[46] Chiara Di Francescomarino et al. “Clustering-based predictive process monitoring”. In: *IEEE transactions on services computing* 12.6 (2016), pp. 896–909 (cit. on pp. 114, 115).

[47] Chiara Di Francescomarino et al. “Predictive process monitoring methods: Which one suits me best?” In: *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16*. Springer. 2018 (cit. on pp. 5, 29, 30, 88, 98, 111).

[48] Remco M. Dijkman et al. “Enabling efficient process mining on large data sets: realizing an in-database process mining operator”. In: *Distributed Parallel Databases* 38.1 (2020), pp. 227–253 (cit. on pp. 9, 19).

[49] John Domingue, Dumitru Roman, and Michael Stollberg. *Web service modeling ontology (WSMO)-An ontology for semantic web services*. 2005 (cit. on p. 31).

[50] Li Dong and Mirella Lapata. “Language to logical form with neural attention”. In: *arXiv preprint arXiv:1601.01280* (2016) (cit. on pp. 21, 22, 24).

[51] Sebastian Dunzer et al. “Conformance checking: a state-of-the-art literature review”. In: *Proceedings of the 11th international conference on subject-oriented business process management*. 2019, pp. 1–10 (cit. on pp. 29, 30, 88, 98).

[52] Marwa Elleuch et al. “Discovering activities from emails based on pattern discovery approach”. In: *International Conference on Business Process Management*. Springer. 2020, pp. 88–104 (cit. on pp. 25, 28).

[53] Marwa Elleuch et al. “Discovering business processes and activities from messaging systems: State-of-the art”. In: *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE. 2020, pp. 137–142 (cit. on p. 25).

[54] Marwa Elleuch et al. “Discovery of activities’ actor perspective from emails based on speech acts detection”. In: *2020 2nd International Conference on Process Mining (ICPM)*. IEEE. 2020, pp. 73–80 (cit. on pp. 25, 28).

[55] Stefan Esser and Dirk Fahland. “Multi-Dimensional Event Data in Graph Databases”. In: *CoRR* abs/2005.14552 (2020) (cit. on pp. 9, 20, 44, 47, 53, 84, 123).

[56] Facebook. *Wit.ai*. `https://wit.ai/`. [Online; accessed 04-March-2022]. 2015 (cit. on p. 23).

[57] Ethan Fast et al. "Iris: A conversational agent for complex tasks". In: *Proceedings of the 2018 CHI conference on human factors in computing systems*. 2018, pp. 1–12 (cit. on p. 23).

[58] Bettina Fazzinga et al. "A framework supporting the analysis of process logs stored in either relational or NoSQL DBMSs". In: *International Symposium on Methodologies for Intelligent Systems*. Springer. 2015, pp. 52–58 (cit. on pp. 4, 18).

[59] Renato César Borges Ferreira, Lucinéia Heloisa Thom, and Marcelo Fantinato. "A Semi-automatic Approach to Identify Business Process Elements in Natural Language Texts." In: *ICEIS (3)*. 2017, pp. 250–261 (cit. on pp. 24, 28).

[60] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. "Process model generation from natural language text". In: *International conference on advanced information systems engineering*. Springer. 2011, pp. 482–496 (cit. on pp. 24, 28, 71).

[61] Anahita Farhang Ghahfarokhi et al. "OCEL: A standard for object-centric event logs". In: *European Conference on Advances in Databases and Information Systems*. Springer. 2021, pp. 169–175 (cit. on pp. 18, 43, 50).

[62] Chih-Wen Goo et al. "Slot-gated modeling for joint slot filling and intent prediction". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. 2018, pp. 753–757 (cit. on p. 23).

[63] Google. *Dialogue flow*. `https://cloud.google.com/dialogflow`. [Online; accessed 04-March-2022]. 2017 (cit. on p. 23).

[64] Archana Goyal, Vishal Gupta, and Manish Kumar. "Recent named entity recognition and classification techniques: a systematic review". In: *Computer Science Review* 29 (2018), pp. 21–43 (cit. on p. 22).

[65] Michael Grohs et al. "Large Language Models can accomplish Business Process Management Tasks". In: *arXiv preprint arXiv:2307.09923* (2023) (cit. on pp. 27, 28).

[66] Martin Gudgin et al. *SOAP Version 1.2*. 2003 (cit. on p. 31).

[67] Christian W Günther and Anne Rozinat. "Disco: Discover Your Processes." In: *BPM (Demos)* (2012) (cit. on pp. 5, 29).

[68] Daniel Guo et al. "Joint semantic utterance classification and slot filling with recursive neural networks". In: *2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2014, pp. 554–559 (cit. on p. 23).

[69] Marc J Hadley. *Web application description language (WADL)*. 2006 (cit. on p. 32).

[70] Xue Han et al. "Bootstrapping natural language querying on process automation data". In: *2020 IEEE International Conference on Services Computing (SCC)*. IEEE. 2020, pp. 170–177 (cit. on pp. 25, 26, 28, 84).

[71] *IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams*. `https://xes-standard.org/`. 2016 (cit. on pp. 18, 41).

[72] Gert Janssenswillen et al. “bupaR: Enabling reproducible business process analysis”. In: *Knowledge-Based Systems* (2019) (cit. on pp. 5, 29, 30).

[73] Mustafa Jarrar and Marios D. Dikaiakos. “Querying the Data Web: The MashQL Approach”. In: *IEEE Internet Comput.* 14.3 (2010), pp. 58–67 (cit. on p. 21).

[74] Urszula Jessen, Michal Sroka, and Dirk Fahland. “Chit-Chat or Deep Talk: Prompt Engineering for Process Mining”. In: *arXiv preprint arXiv:2307.09909* (2023) (cit. on pp. 27, 28).

[75] Changjiu Jin et al. “GBLENDER: towards blending visual query formulation and query processing in graph databases”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 2010, pp. 111–122 (cit. on p. 21).

[76] Tao Jin et al. “Efficient querying of large process model repositories”. In: *Comput. Ind.* 64.1 (2013), pp. 41–49 (cit. on p. 18).

[77] Edward Junprung. “Exploring the Intersection of Large Language Models and Agent-Based Modeling via Prompt Engineering”. In: *arXiv preprint arXiv:2308.07411* (2023) (cit. on p. 27).

[78] Klaus Kammerer, Jens Kolb, and Manfred Reichert. “PQL-a descriptive language for querying, abstracting and changing process models”. In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2015, pp. 135–150 (cit. on p. 18).

[79] Aikaterini Karavisileiou, Nikolaos Mainas, and Euripides GM Petrakis. “Ontology for openapi rest services descriptions”. In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2020, pp. 35–40 (cit. on p. 32).

[80] Christoph Kecht et al. “Event log construction from customer service conversations using natural language inference”. In: *2021 3rd International Conference on Process Mining (ICPM)*. IEEE. 2021, pp. 144–151 (cit. on p. 25).

[81] Uwe Keller et al. “Wsmo web service discovery”. In: *WSML Working Draft D* 5 (2004) (cit. on pp. 31, 33).

[82] Anita Khatri and OP Rishi. “Augmenting cloud service discovery using ontology”. In: *Rising Threats in Expert Applications and Solutions: Proceedings of FICR-TEAS 2020*. Springer. 2021, pp. 193–201 (cit. on p. 32).

[83] Yoon Kim. “Convolutional neural networks for sentence classification”. In: *arXiv preprint arXiv:1408.5882* (2014) (cit. on p. 22).

[84] Matthias Klusch, Benedikt Fries, and Katia Sycara. “Automated semantic web service discovery with OWLS-MX”. In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 915–922 (cit. on p. 33).

[85] Meriana Kobeissi et al. “An intent-based natural language interface for querying process execution data”. In: *2021 3rd International Conference on Process Mining (ICPM)*. IEEE. 2021, pp. 152–159 (cit. on p. 57).

[86] Meriana Kobeissi et al. “Natural language querying of process execution data”. In: *Information Systems* 116 (2023), p. 102227 (cit. on p. 57).

[87] Takeshi Kojima et al. “Large language models are zero-shot reasoners”. In: *Advances in neural information processing systems* 35 (2022), pp. 22199–22213 (cit. on p. 27).

[88] Gakuto Kurata et al. “Leveraging sentence-level information with encoder lstm for semantic slot filling”. In: *arXiv preprint arXiv:1601.01530* (2016) (cit. on p. 23).

[89] Kabul Kurniawan, Fajar J Ekaputra, and Peb R Aryan. “Semantic service description and compositions: A systematic literature review”. In: *2018 2nd International Conference on Informatics and Computational Sciences (ICICoS)*. IEEE. 2018, pp. 1–6 (cit. on pp. 31, 32).

[90] Marcello La Rosa et al. “APROMORE: An advanced process model repository”. In: *Expert Systems with Applications* (2011) (cit. on pp. 5, 29, 30).

[91] Nassim Laga et al. “Emails Analysis for Business Process Discovery.” In: *ATAED@ Petri Nets/ACSD*. 2019, pp. 54–70 (cit. on pp. 25, 28).

[92] Sven Lambrechts, Ir WMP van der Aalst, and AJMM Weijters. “Scenario-based process mining: Web servicing and automated scenario generation”. PhD thesis. Technische Universiteit Eindhoven, Eindhoven, The Netherlands, 2009 (cit. on pp. 29, 30).

[93] Sana Ben Abdallah Ben Lamine et al. “An ontology-based approach for personalized RESTful Web service discovery”. In: *Procedia computer science* 112 (2017), pp. 2127–2136 (cit. on p. 33).

[94] Suk Kyoon Lee and Kyu-Young Whang. “VOQL*: A Visual Object Query Language With Inductively Defined Formal Semantics”. In: *J. Vis. Lang. Comput.* 12.4 (2001), pp. 413–433 (cit. on p. 21).

[95] Sander JJ Leemans, Dirk Fahland, and Wil MP Van Der Aalst. “Discovering block-structured process models from event logs-a constructive approach”. In: *Application and Theory of Petri Nets and Concurrency: 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings 34*. Springer. 2013, pp. 311–329 (cit. on p. 95).

[96] Anna Leontjeva et al. “Complex symbolic sequence encodings for predictive monitoring of business processes”. In: *Business Process Management: 13th International Conference, BPM 2015, Innsbruck, Austria, August 31–September 3, 2015, Proceedings 13*. Springer. 2015, pp. 297–313 (cit. on pp. 114, 115).

[97] Jing Li et al. “A Survey on Deep Learning for Named Entity Recognition”. In: *IEEE Trans. Knowl. Data Eng.* 34.1 (2022), pp. 50–70 (cit. on p. 22).

[98] Bing Liu and Ian Lane. “Attention-based recurrent neural network models for joint intent detection and slot filling”. In: *arXiv preprint arXiv:1609.01454* (2016) (cit. on p. 23).

[99] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. “Adversarial multi-task learning for text classification”. In: *arXiv preprint arXiv:1704.05742* (2017) (cit. on p. 23).

[100] Ignacio Lizarralde et al. “Exploiting named entity recognition for improving syntactic-based web service discovery”. In: *Journal of Information Science* 45.3 (2019), pp. 398–415 (cit. on p. 32).

[101] Jorge Lloret-Gazo. “A Survey on Visual Query Systems in the Web Era (extended version)”. In: *CoRR* abs/1708.00192 (2017) (cit. on p. 21).

[102] J Mannar Mannan, Murugesan Sundarambal, and S Raghul. “Selection of Ontology for Web Service Description Language to Ontology Web Language conversion.” In: *J. Comput. Sci.* 10.1 (2014), pp. 45–53 (cit. on p. 31).

[103] Ronny Mans, Wil MP van der Aalst, and HMW Verbeek. “Supporting Process Mining Workflows with RapidProM.” In: *BPM (Demos)* (2014) (cit. on p. 5).

[104] Sunilkumar S Manvi and Gopal Krishna Shyam. “Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey”. In: *Journal of network and computer applications* 41 (2014), pp. 424–440 (cit. on p. 31).

[105] Alfonso E Márquez-Chamorro et al. “Run-time prediction of business process indicators using evolutionary decision rules”. In: *Expert Systems with Applications* 87 (2017), pp. 1–14 (cit. on p. 115).

[106] Alfonso Eduardo Márquez-Chamorro, Manuel Resinas, and Antonio Ruiz-Cortés. “Predictive monitoring of business processes: a survey”. In: *IEEE Transactions on Services Computing* (2017) (cit. on pp. 29, 30, 88, 98, 111).

[107] David Martin et al. “OWL-S: Semantic markup for web services”. In: *W3C member submission* 22.4 (2004) (cit. on pp. 31, 33).

[108] Mark Masse. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O’Reilly Media, Inc.", 2011 (cit. on p. 92).

[109] Deborah L McGuinness, Frank Van Harmelen, et al. “OWL web ontology language overview”. In: *W3C recommendation* 10.10 (2004), p. 2004 (cit. on p. 31).

[110] Sheila A McIlraith, Tran Cao Son, and Honglei Zeng. “Semantic web services”. In: *IEEE intelligent systems* 16.2 (2001), pp. 46–53 (cit. on p. 31).

[111] Sonia Ben Mokhtar et al. “Interoperable semantic and syntactic service discovery for ambient computing environments”. In: *International Journal of Ambient Computing and Intelligence (IJACI)* 2.4 (2010), pp. 13–32 (cit. on p. 32).

[112] Mariusz Momotko and Kazimierz Subieta. “Process Query Language: A Way to Make Workflow Processes More Flexible”. In: *Advances in Databases and Information Systems, 8th East European Conference, ADBIS*. Vol. 3255. 2004, pp. 306–321 (cit. on p. 18).

[113] Eduardo González López de Murillas, Hajo A. Reijers, and Wil M. P. van der Aalst. “Connecting databases with process mining: a meta model and toolset”. In: *Softw. Syst. Model.* 18.2 (2019), pp. 1209–1247 (cit. on pp. 9, 19, 43).

[114] Le Duy Ngan and Rajaraman Kanagasabai. “Semantic Web service discovery: state-of-the-art and research challenges”. In: *Personal and ubiquitous computing* 17 (2013), pp. 1741–1752 (cit. on p. 32).

[115] Nils Agne Nordbotten. “XML and web services security standards”. In: *IEEE Communications Surveys & Tutorials* 11.3 (2009), pp. 4–21 (cit. on p. 31).

[116] Aabhas V Paliwal et al. “Semantics-based automated service discovery”. In: *IEEE Transactions on Services Computing* 5.2 (2011), pp. 260–275 (cit. on p. 32).

[117] Dhruv Patel, Harsh Anand, and Suchetana Chakraborty. “CrossTrustchain: Cross-Chain Interoperability using Multivariate Trust Models”. In: *2023 15th International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE. 2023, pp. 129–134 (cit. on p. 32).

[118] Anastasiia Pika et al. “Predicting deadline transgressions using event logs”. In: *Business Process Management Workshops: BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers 10*. Springer. 2013, pp. 211–216 (cit. on p. 115).

[119] Mirko Polato et al. “Time and activity sequence prediction of business process instances”. In: *Computing* 100 (2018), pp. 1005–1031 (cit. on p. 115).

[120] Artem Polyvyanyy, Marcello La Rosa, and Arthur HM Ter Hofstede. “Indexing and efficient instance-based retrieval of process models using untanglings”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2014, pp. 439–456 (cit. on p. 18).

[121] Artem Polyvyanyy et al. “Process querying: Enabling business intelligence through query-based process analytics”. In: *Decision Support Systems* 100 (2017), pp. 41–56 (cit. on pp. 2, 3, 56, 62).

[122] Artem Polyvyanyy et al. “Process querying: Enabling business intelligence through query-based process analytics”. In: *Decis. Support Syst.* 100 (2017), pp. 41–56 (cit. on p. 18).

[123] Elizaveta Povalyaeva, Ismail Khamitov, and Artyom Fomenko. “Bpic 2017: density analysis of the interaction with clients”. In: *BPI Challenge* (2017) (cit. on p. 50).

[124] Margus Räim et al. “Log-Based Understanding of Business Processes through Temporal Logic Query Checking”. In: *On the Move to Meaningful Internet Systems: OTM*. Vol. 8841. 2014, pp. 75–92 (cit. on pp. 9, 18, 41).

[125] Jinghai Rao, Peep Kungas, and Mihhail Matskin. “Logic-based web services composition: From service description to process model”. In: *Proceedings. IEEE International Conference on Web Services, 2004.* IEEE. 2004, pp. 446–453 (cit. on p. 33).

[126] Pushpendre Rastogi et al. “Scaling multi-domain dialogue state tracking via query reformulation”. In: *arXiv preprint arXiv:1903.05164* (2019) (cit. on p. 23).

[127] Suman Ravuri and Andreas Stolcke. “Recurrent neural network and LSTM models for lexical utterance classification”. In: *Sixteenth annual conference of the international speech communication association*. 2015 (cit. on p. 22).

[128] Jie Ren and Jun Shen. “Research on the Description Method of the Atomic Services in Extensible Network Service Model”. In: *Geo-Spatial Knowledge and Intelligence: 4th International Conference on Geo-Informatics in Resource Management and Sustainable Ecosystem, GRMSE 2016, Hong Kong, China, November 18-20, 2016, Revised Selected Papers, Part II 4*. Springer. 2017, pp. 191–199 (cit. on p. 32).

[129] Manuel Resinas, Adela del Río-Ortega, and Han van der Aa. "From Text to Performance Measurement: Automatically Computing Process Performance Using Textual Descriptions and Event Logs". In: *International Conference on Business Process Management*. Springer. 2023, pp. 266–283 (cit. on pp. 26, 28).

[130] *Resource Description Framework (RDF): Concepts and Abstract Syntax*. `https://www.w3.org/TR/rdf-concepts/`. 2004 (cit. on pp. 19, 44).

[131] Aleksandra Revina and Ünal Aksu. "An approach for analyzing business process execution complexity based on textual data and event log". In: *Information Systems* 114 (2023), p. 102184 (cit. on p. 25).

[132] Francesco Riva et al. "An SQL-Based Declarative Process Mining Framework for Analyzing Process Data Stored in Relational Databases". In: *International Conference on Business Process Management*. Springer. 2023, pp. 214–231 (cit. on p. 19).

[133] Andreas Rogge-Solti and Mathias Weske. "Prediction of business process durations using non-Markovian stochastic Petri nets". In: *Information Systems* 54 (2015), pp. 1–14 (cit. on pp. 109, 115).

[134] Eric Rojas et al. "Process mining in healthcare: A literature review". In: *Journal of biomedical informatics* 61 (2016), pp. 224–236 (cit. on pp. 28, 30).

[135] Diptikalyan Saha et al. "ATHENA: An Ontology-Driven System for Natural Language Querying over Relational Data Stores". In: *Proc. VLDB Endow.* (2016) (cit. on pp. 21, 22, 24, 26).

[136] Maria-Isabel Sanchez-Segura et al. "Valuable Business Knowledge Asset Discovery by Processing Unstructured Data". In: *Sustainability* 14.20 (2022), p. 12971 (cit. on p. 25).

[137] Cleiton dos Santos Garcia et al. "Process mining techniques and applications–A systematic mapping study". In: *Expert Systems with Applications* 133 (2019), pp. 260–295 (cit. on pp. 28, 30).

[138] Joachim Schaper. "Cloud Services". In: *4th IEEE International Conference on Digital Ecosystems and Technologies*. IEEE. 2010, pp. 91–91 (cit. on p. 31).

[139] Stefan Schönig et al. "Efficient and Customisable Declarative Process Mining with SQL". In: *Advanced Information Systems Engineering - 28th International Conference, CAiSE*. Vol. 9694. 2016, pp. 290–305 (cit. on pp. 9, 19, 43).

[140] Daniel Schuster, Sebastiaan J van Zelst, and Wil MP van der Aalst. "Utilizing domain knowledge in data-driven process discovery: A literature review". In: *Computers in Industry* 137 (2022), p. 103612 (cit. on pp. 10, 29, 30, 97, 98).

[141] Ali Shahaab et al. "A hybrid blockchain implementation to ensure data integrity and interoperability for public service organisations". In: *2021 IEEE International Conference on Blockchain (Blockchain)*. IEEE. 2021, pp. 295–305 (cit. on p. 32).

[142] Saeedeh Shekarpour et al. "SINA: Semantic interpretation of user queries for question answering on interlinked data". In: *J. Web Semant.* 30 (2015), pp. 39–51 (cit. on p. 22).

[143] Amit P Sheth, Karthik Gomadam, and Jon Lathem. "SA-REST: Semantically interoperable and easier-to-use services and mashups". In: *IEEE Internet Computing* 11.6 (2007), pp. 91–94 (cit. on p. 32).

[144] Alan Sill. "The design and architecture of microservices". In: *IEEE Cloud Computing* 3.5 (2016), pp. 76–80 (cit. on p. 32).

[145] Renuka Sindhgatta, Aditya Ghose, and Hoa Khanh Dam. "Leveraging unstructured data to analyze implicit process context". In: *Business Process Management Forum: BPM Forum 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings 16*. Springer. 2018, pp. 143–158 (cit. on p. 25).

[146] Liu Sipei et al. "Description logic rule and process algebra based OWL-S modeling, matching and composition". In: *2011 IEEE Consumer Communications and Networking Conference (CCNC)*. IEEE. 2011, pp. 153–157 (cit. on p. 33).

[147] Tijs Slaats. "Declarative and hybrid process discovery: Recent advances and open challenges". In: *Journal on Data Semantics* 9.1 (2020), pp. 3–20 (cit. on pp. 10, 29, 30, 97).

[148] Dezhao Song et al. "TR discover: A natural language interface for querying and analyzing interlinked datasets". In: *International Semantic Web Conference*. Springer. 2015, pp. 21–37 (cit. on pp. 21, 24).

[149] Christina Sun. *A Natural Language Interface for Querying Graph Databases*. Master Report. Massachusetts Institute of Technology, 2018 (cit. on p. 22).

[150] Vijay Surwase. "REST API modeling languages-a developer's perspective". In: *Int. J. Sci. Technol. Eng* 2.10 (2016), pp. 634–637 (cit. on p. 32).

[151] Zhixing Tan et al. "THUMT: An Open-Source Toolkit for Neural Machine Translation". In: *Proceedings of the 14th Conference of the Association for Machine Translation in the Americas, AMTA*. 2020, pp. 116–122 (cit. on pp. 21, 22, 24).

[152] Niek Tax et al. "Predictive business process monitoring with LSTM neural networks". In: *Advanced Information Systems Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings 29*. Springer. 2017, pp. 477–492 (cit. on pp. 114, 115).

[153] Irene Teinemaa et al. "Predictive business process monitoring with structured and unstructured data". In: *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings 14*. Springer. 2016, pp. 401–417 (cit. on p. 25).

[154] Ashutosh Tiwari, Chris J Turner, and Basim Majeed. "A review of business process mining: state-of-the-art and future trends". In: *Business Process Management Journal* 14.1 (2008), pp. 5–22 (cit. on pp. 28, 30).

[155] WeiTek Tsai, XiaoYing Bai, and Yu Huang. "Software-as-a-service (SaaS): perspectives and challenges". In: *Science China Information Sciences* 57 (2014), pp. 1–15 (cit. on p. 31).

[156] Wil Van Der Aalst and Wil van der Aalst. *Data science in action*. Springer, 2016 (cit. on pp. 4, 56, 62, 87, 89).

[157] Boudewijn F Van Dongen et al. “The ProM framework: A new era in process mining tool support”. In: *Applications and Theory of Petri Nets 2005: 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005. Proceedings 26*. Springer. 2005 (cit. on pp. 5, 29).

[158] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 23).

[159] Ilya Verenich et al. “Minimizing overprocessing waste in business processes via predictive activity ordering”. In: *Advanced Information Systems Engineering: 28th International Conference, CAiSE 2016, Ljubljana, Slovenia, June 13-17, 2016. Proceedings 28*. Springer. 2016, pp. 186–202 (cit. on pp. 114, 115).

[160] Ngoc Thang Vu. “Sequential convolutional neural networks for slot filling in spoken language understanding”. In: *arXiv preprint arXiv:1606.07783* (2016) (cit. on p. 23).

[161] Michael Werner and Nick Gehrke. “Multilevel process mining for financial audits”. In: *IEEE Transactions on Services Computing* 8.6 (2015), pp. 820–832 (cit. on p. 19).

[162] Erik Wilde and Cesare Pautasso. *REST: from research to practice*. Springer Science & Business Media, 2011 (cit. on p. 31).

[163] Puyang Xu and Ruhi Sarikaya. “Convolutional neural network based triangular crf for joint intent detection and slot filling”. In: *2013 ieee workshop on automatic speech recognition and understanding*. IEEE. 2013, pp. 78–83 (cit. on p. 23).

[164] Zichao Yang et al. “Hierarchical attention networks for document classification”. In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489 (cit. on p. 23).

[165] Kaisheng Yao et al. “Spoken language understanding using long short-term memory neural networks”. In: *2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2014, pp. 189–194 (cit. on p. 23).

[166] Robail Yasrab. “Platform-as-a-service (paas): the next hype of cloud computing”. In: *arXiv preprint arXiv:1804.10811* (2018) (cit. on p. 31).

[167] Peipei Yi et al. “AutoG: a visual query autocompletion framework for graph databases”. In: *VLDB J.* 26.3 (2017), pp. 347–372 (cit. on p. 21).

[168] Karn Yongsiriwit, Nguyen Ngoc Chan, and Walid Gaaloul. “Log-Based Process Fragment Querying to Support Process Design”. In: *48th Hawaii International Conference on System Sciences, HICSS*. 2015, pp. 4109–4119 (cit. on pp. 4, 9, 18, 41).

[169] Lei Yu et al. “A web services description language-based description model of internet of things services”. In: *Sensor Letters* 12.2 (2014), pp. 448–455 (cit. on p. 31).

[170] Pierluigi Zerbino, Alessandro Stefanini, and Davide Aloini. “Process science in action: A literature review on process mining in business management”. In: *Technological Forecasting and Social Change* 172 (2021), p. 121021 (cit. on p. 28).

[171] Furkh Zeshan, Radziah Mohamad, and MN Ahmad. “Semantic web service composition approaches: overview and limitations”. In: *International Journal on New Computer Architectures and Their Applications (IJNCAA)* 1.3 (2011), pp. 640–651 (cit. on p. 31).

[172] Neng Zhang et al. “WSGM-SD: an approach to RESTful service discovery based on weighted service goal model”. In: *Chinese Journal of Electronics* 25.2 (2016), pp. 256–263 (cit. on p. 33).

[173] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems* 28 (2015) (cit. on p. 22).

[174] Zhiwei Zhao and Youzheng Wu. “Attention-Based Convolutional Neural Networks for Sentence Classification.” In: *Interspeech*. Vol. 8. 2016, pp. 705–709 (cit. on p. 22).

[175] Weiguo Zheng et al. “Natural Language Question/Answering: Let Users Talk With The Knowledge Graph”. In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM*. 2017, pp. 217–226 (cit. on pp. 21, 22, 24).

[176] Zhiling Zheng et al. “ChatGPT Chemistry Assistant for Text Mining and Prediction of MOF Synthesis”. In: *arXiv preprint arXiv:2306.11296* (2023) (cit. on p. 27).

[177] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning”. In: *CoRR* abs/1709.00103 (2017) (cit. on pp. 21, 22, 24).

[178] Neli Zlatareva and Devansh Amin. “Natural Language to SPARQL Query Builder for Semantic Web Applications”. In: *Journal of Machine Intelligence and Data Science (JMIDS)* 2 (2021) (cit. on p. 34).

DOCTORAL SCHOOL

Titre: Un cadre d'IA conversationnelle pour l'analyse cognitive des processus

Mots clés: Analyse de Processus, Interrogation de Données de Processus, Exploration de Processus, Services Cognitifs, Architecture Orientée Services

Résumé: Les processus métier constituent les piliers fondamentaux des organisations, englobant toute une gamme d'activités structurées visant à atteindre des objectifs organisationnels distincts. Ces processus, caractérisés par une multitude de tâches, d'interactions et de flux de travail, offrent une méthodologie structurée pour superviser les opérations cruciales dans divers secteurs. Une découverte essentielle pour les organisations a été la reconnaissance de la valeur profonde inhérente aux données produites pendant ces processus. L'analyse des processus, une discipline spécialisée, explore ces journaux de données, facilitant une compréhension plus profonde et l'amélioration des BP. Cette analyse peut être catégorisée en deux perspectives : le niveau d'instance, qui se concentre sur les exécutions individuelles de processus, et le niveau de processus, qui examine le processus global. Cependant, l'application de l'analyse des processus pose des défis aux utilisateurs, impliquant la nécessité d'accéder aux données, de naviguer dans les API de bas niveau et d'utiliser des méthodes dépendantes d'outils. L'application dans le monde réel rencontre souvent des complexités et des obstacles centrés sur l'utilisateur. Plus précisément, l'analyse de niveau d'instance exige des utilisateurs qu'ils accèdent aux données d'exécution de processus stockées, une tâche qui peut être complexe pour les professionnels de l'entreprise en raison de l'exigence de maîtriser des langages de requête complexes tels que SQL et CYPHER. En revanche, l'analyse de niveau de processus des données de processus implique l'utilisation de méthodes et d'algorithmes qui exploitent les données d'exécution de processus extraites des systèmes d'information. Ces méthodologies sont regroupées sous le terme de techniques d'exploration de processus. L'application de l'exploration de processus confronte les analystes à la tâche complexe de sélection de méthodes, qui consiste à trier des descriptions de méthodes non structurées. De plus, l'application des méthodes d'exploration de processus dépend d'outils spécifiques et nécessite un certain niveau d'expertise technique.

Pour relever ces défis, cette thèse présente des solutions basées sur l'IA, mettant l'accent sur l'intégration de capacités cognitives dans l'analyse des processus pour faciliter les tâches d'analyse tant au niveau de l'instance qu'au niveau du processus pour tous les utilisateurs. Les objectifs principaux sont doubles : premièrement, améliorer l'accessibilité des données d'exécution de processus en créant une interface capable de construire automatiquement la requête de base correspondante à partir du langage naturel. Ceci est complété par la proposition d'une technique de stockage adaptée et d'un langage de requête autour desquels l'interface doit être conçue. À cet égard, nous introduisons un méta-modèle graphique basé sur le graphe de propriétés étiquetées (LPG) pour le stockage efficace des données. Deuxièmement, pour rationaliser la découverte et l'accessibilité des techniques d'exploration de processus, nous présentons une architecture orientée services. Cette architecture comprend trois composants principaux : un méta-modèle LPG détaillant les méthodes d'exploration de processus, une conception orientée services REST adaptée à ces méthodes, et un composant habile à mettre en correspondance les besoins des utilisateurs exprimés en langage naturel avec les services appropriés.
Pour valider notre méta-modèle graphique, nous avons utilisé deux ensembles de données de processus accessibles au public disponibles à la fois au format CSV et OCEL. Ces ensembles de données ont été essentiels pour évaluer les performances de notre pipeline de requêtes en langage naturel. Nous avons recueilli des requêtes en langage naturel auprès d'utilisateurs externes et en avons généré d'autres à l'aide d'outils de paraphrase. Notre cadre orienté services a été évalué à l'aide de requêtes en langage naturel spécialement conçues pour les descriptions de services d'exploration de processus. De plus, nous avons mené une étude de cas avec des participants externes pour évaluer l'expérience utilisateur et recueillir des commentaires. Nous fournissons publiquement les résultats de l'évaluation pour garantir la reproductibilité dans le domaine étudié.

Title: A Conversational AI framework for cognitive process analysis

Keywords: Process Analysis, Process Data Querying, Process Mining, Cognitive Services, Service-Oriented Architecture

Abstract: Business processes (BP) are the foundational pillars of organizations, encapsulating a range of structured activities aimed at fulfilling distinct organizational objectives. These processes, characterized by a plethora of tasks, interactions, and workflows, offer a structured methodology for overseeing crucial operations across diverse sectors. A pivotal insight for organizations has been the discernment of the profound value inherent in the data produced during these processes. Process analysis, a specialized discipline, ventures into these data logs, facilitating a deeper comprehension and enhancement of BPs. This analysis can be categorized into two perspectives: instance-level, which focuses on individual process executions, and process-level, which examines the overarching process. However, applying process analysis in practice poses challenges for users, involving the need to access data, navigate low-level APIs, and employ tool-dependent methods. Real-world application often encounters complexities and user-centric obstacles.

Specifically, instance-level analysis demands users to access stored process execution data, a task that can be intricate for business professionals due to the requirement of mastering complex query languages like SQL and CYPHER. Conversely, process-level analysis of process data involves the utilization of methods and algorithms that harness process execution data extracted from information systems. These methodologies collectively fall under the umbrella of process mining techniques. The application of process mining confronts analysts with the intricate task of method selection, which involves sifting through unstructured method descriptions. Additionally, the application of process mining methods depends on specific tools and necessitates a certain level of technical expertise.

To address these challenges, this thesis introduces AI-driven solutions, with a focus on integrating cognitive capabilities into process analysis to facilitate analysis tasks at both the instance level and the process level for all users. The primary objectives are twofold: Firstly, to enhance the accessibility of process execution data by creating an interface capable of automatically constructing the corresponding database query from natural language. This is complemented by proposing a suitable storage technique and query language that the interface should be designed around. In this regard, we introduce a graph metamodel based on Labeled Property Graph (LPG) for efficient data storage. Secondly, to streamline the discovery and accessibility of process mining techniques, we present a service-oriented architecture. This architecture comprises three core components: an LPG meta-model detailing process mining methods, a service-oriented REST API design tailored for these methods, and a component adept at matching user requirements expressed in natural language with appropriate services.

For the validation of our graph metamodel, we utilized two publicly accessible process datasets available in both CSV and OCEL formats. These datasets were instrumental in evaluating the performance of our NL querying pipeline. We gathered NL queries from external users and produced additional ones through paraphrasing tools. Our service-oriented framework underwent an assessment using NL queries specifically designed for process mining service descriptions. Additionally, we carried out a use case study with external participants to evaluate user experience and to gather feedback. We publically provide the evaluation results to ensure reproducibility in the studied area.

Institut Polytechnique de Paris

91120 Palaiseau, France