# An Intent-Based Natural Language Interface for Querying Process Execution Data

Meriana Kobeissi *, Nour Assy *, Walid Gaaloul *, Bruno Defude*, Bassem Haidar †
*Telecom SudParis, Institut Polytechnique de Paris, France, email : {*FirstName*}.{*LastName*}@*telecom-sudparis.eu*
† Lebanese University, Faculty of Information, Beirut, Lebanon, email : {*FirstName*}.{*LastName*}@*ul.edu.lb*

***Abstract*—Process mining techniques allow organizations to discover, monitor and improve their as-is processes by analyzing the process execution data, aka event data, recorded by their information systems. A recurrent task in process mining is *querying*. Querying allows users to get insights into specific executions of their processes and to retrieve relevant data. Existing process querying techniques require end users to be knowledgeable of the query language and the database schema. However, a key success factor for process analysis is to make querying accessible to business experts who may be inexperienced in database querying. This paper addresses this challenge by proposing a natural language interface (NLI) for querying event data. The interface allows users to formulate their questions in natural language and to automatically translate the questions into a structured query that can be executed over a database. We use graph based storage techniques, namely labeled property graphs, which allow to explicitly model event data relationships. As an executable query language, we use the Cypher language which is widely used for querying property graphs. The approach has been implemented and evaluated using a publicly available event log.**

***Index Terms*—Process Querying, Process Mining, Natural Language Interface, graph database, Cypher language**

## I. Introduction

The increasing availability of event data has created novel opportunities for analyzing and extracting knowledge from large data sets. Under the umbrella of process analytics [5], different process-oriented data analysis techniques have been developed to help organizations get data-driven insights about the execution of their processes. These techniques are applied over the traces of event data which consist of interrelated events recorded by the organizations' information systems. Each event records the *activity* that was executed, the *time* at which it was executed, the *data object or artifact* affected by the activity and any other related information such as the *resources* that performed the activity. For instance, using process mining techniques [1], analysts can answer questions related to how the process is performing, where deviations and bottlenecks occur and what actions are recommended.

The majority of process analysis techniques seek to extract patterns that can generalize the observed behavior. However, in many situations, process analysts need to answer questions at a fine-grained level with the intent of having immediate glimpse on some aspects of the process execution. Examples include: *Who validated application #10? When is the interview scheduled for Candidate C1? Give me all information related to the processing of application 15.*

Different approaches have been proposed to support the querying of event data which are stored in either relational or graphical models (for an overview, refer to [13]). All existing process querying techniques require end users to master, not only the query language specific to the data storage technology used, but also the extensions of the language that are proposed to query event data. However, process querying is primarily business-driven and should be accessible to business experts who may not be aware of the structure of the database nor familiar with database querying. In this paper, we propose a natural language interface (NLI) that assists end users in querying event data. The interface takes a plain text question, automatically constructs the corresponding query to be executed over the database, and returns the response. We divided the queries into three categories based on the type of information they return (structural, behavioral, and performance queries). In this paper, we focus on how to build structural queries, i.e. queries that answer questions about event data.

Compared to existing works, the main contributions of this paper can be summarized as follow. First, we propose a NLI system to query graph databases based on labeled property graphs [3] and using the Cypher language.[1] Our choice is motivated by the fact that process querying comes with an intrinsic need for relationship analysis. Since, graph databases model data relationships explicitly, they are considered more efficient in querying paths. In addition, their query languages have the desired expressivity especially for path queries.

Second, our proposed NLI system is hybrid and takes advantages of both machine-learning and rule-based approaches. The approach consists of two main components. The first machine-learning based component performs Natural Language Understanding (NLU) which consists of intent detection and named entity recognition (NER). The second rule-based component constructs the corresponding Cypher query. The experiments performed on a real-life event log from BPI challenge'17 [10] show that our intent-based approach increases the accuracy of the NLI system.

The remainder of this paper is organized as follows. In Section II, we discuss the existing works related to our contribution. In section III, we define some concepts related to event data storage and we briefly present the Cypher graph query language. The approach is presented in section IV. Finally, in

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


2021 3rd International Conference on Process Mining (ICPM 2021), available at https://doi.org/10.1109/ICPM53251.2021

Section V, we present and discuss the evaluation results before concluding and discussing future works in Section VI.

## II. Related Work

Our work is related to two major research areas: process querying and NLIs for databases.

### A. Process Querying

Existing works on process querying can be classified according to the data input and the querying goal [14]. Our work falls into the category which focuses on querying execution traces of business processes in the form of event logs.

Authors in [15], [21] propose to query event data in the form of XES event logs.[2] The XES standard groups events under a single case notion and does not allow to model multi-dimensional process data. Different approaches are proposed to query process logs stored in relational databases [8], [12], [17]. While relational data modeling is simple and widely used, querying the behavioral aspect is complicated using SQL since the relationships between events are not explicitly stored. The authors in [4] propose an extension of SPARQL that allows them to query event data stored in RDF graphs.[3]

Our data storage model is closely related to [11] in which multi-dimensional event data are stored in a labeled property graph and queried using the Cypher language. The main difference is that our proposed model is closely related to OCEL standard. All existing process querying techniques propose data models or query language extensions. Our work has a different objective. We propose a NLI to facilitate the querying task and make it accessible to a wide range of users.

### B. Natural Language Interfaces

Our work is closely related to natural language interfaces for querying databases (NLIDB). NLIDB systems interpret the question in natural language, then construct the corresponding query to be executed in order to return the desired response. Existing works can be categorized into rule-based or deep-learning based approaches. The authors in [2] review most recent rule-based approaches and provide a comparison based on the techniques used and the questions that can be answered. These approaches provide a high level of flexibility and adaptability to new databases. The main limitation of rule-based methods is their limited scope since they make several assumptions about the database, the query, or the natural language question.

Recently, deep-learning based methods have been proposed. These approaches employ sequence to sequence models that translate natural language queries to SQL queries [9], [20], [23]. The basic idea is to employ a machine learning model that takes the natural language question as input and attempts to predict the translated SQL query based on observed patterns. The main advantage of machine learning-based approaches over traditional NLIs is that they support linguistic diversity, giving the user more flexibility in formulating their questions. However, one of the most significant challenges in developing these methods is the lack of training data. Moreover, these systems act as a black box, preventing the user from knowing whether the failure occurred due to linguistic issues in the question or because the database does not contain the desired result. In this paper, we propose a hybrid NLI system that combines rule-based and machine learning approaches.

The majority of current NLI systems are proposed for querying relational databases [6], [7], [16] or RDF graphs [18], [22]. Our work is related to [19] in which an approach is proposed to construct a structured subgraph from natural language. However, this work is limited to the construction of a subgraph and does not address the problem of converting the subgraph into a query. Moreover, in our work we employ an intent detection step which increases the system accuracy.

## III. Preliminaries

### A. Graph-based storage of event data

This section presents our proposed graph metamodel to store event data based on labeled property graphs [3]. The proposed metamodel is closely related to OCEL,[4] a standard that has been recently proposed for storing object-centric event data that do not enforce a specific case notion.

A labeled property graph is a directed labeled multigraph consisting of labeled nodes and relations. Each node/relation may have properties which correspond to key-value pairs of attributes. A formal definition of a labeled property graph is given in Definition 1. We assume that $L$ is a set of labels (for nodes and edges), $P$ 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 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 a label 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 $V$.*

Fig.1 shows our proposed event property graph metamodel based on property graphs. An example of its instantiation populated with loan application data is illustrated in Fig. 2.

The metamodel in Fig. 1 describes the labels of nodes and edges, the allowed relations and the minimal set of properties required to store event data. Nodes in the graph refer to the various types of information that can be extracted from event data. We distinguish the mandatory node labels: *Activity* which refers to an activity instance executed within an event; and nodes that refer to artifact objects manipulated by activities

[2]Extensible Event Stream: www.xes-standard.org

[3]Resource Description Framework: www.w3.org/TR/rdf-concepts/

[4]OCEL standart: http://ocel-standard.org/

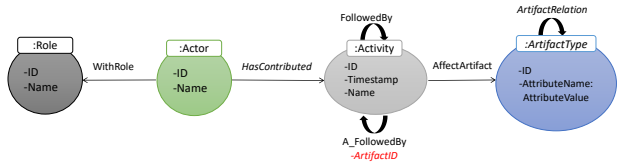


Fig. 1: Event Property Graph Metamodel

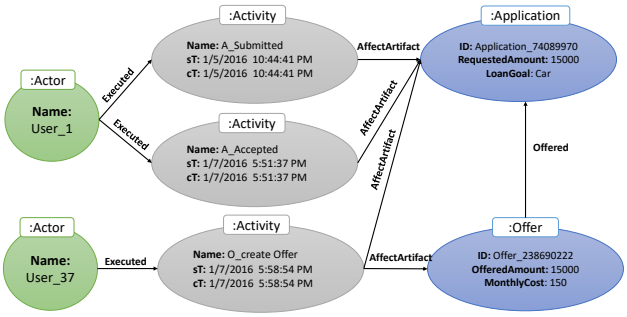


Fig. 2: An example of an event property graph

and that are labeled with the corresponding artifact type. The activity node has the activity instance id, name and timestamp as mandatory properties. The Activity node is connected to the Artifact node through the relation *AffectArtifact* which indicates that the corresponding object is involved in the execution of the activity.

To model the temporal order between activity instances, we distinguish two types of relations: *FollowedBy* and *A_FollowedBy*. The first relation allows to model the temporal order between all activity instances in the graph. The second, allows to model the temporal order between activity instances from the perspective of a specific artifact object. Artifact nodes may also be connected through the relation *ArtifactRelation* which is replaced by the actual relation name.

In addition to Activity and Artifact node types, the *Actor* and *Role* nodes can be created in case information about the resources is available. The Actor node is connected to the Activity node through the relation *HasContributed* which indicates the type of involvement (e.g. executor) of a specific actor in the execution of the activity. Similarly the *Role* node indicates specific roles attributed to actors.

### B. The Cypher language

To query the property graph, we use the Cypher language, 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. Our proposed NLI supports the construction of Cypher queries with their main clauses: *MATCH-WHERE-WITH-ORDER BY-RETURN*.

- **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/edges 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/edges or properties' values) as well as sub-graphs or any defined variables inside the query.

An example of a Cypher query that shows all different clauses is shown in Table III. The query answers the question given in the first row of the table.

## IV. Intent-based NLI for process data querying

In this section, we describe our NLI system to assist end users in querying event data stored in an event property graph. The approach takes as input a question expressed in plain text and returns as output the answer retrieved from the graph DB. The approach consists of two main components: (i) intent detection and entities extraction (detailed in Section IV-A), and (ii) query construction (detailed in Section IV-B). The first component allows the system to understand the question intent, i.e. what does the user want and subsequently what should be searched for in the graph DB. It also allows to extract the main entities from the question that will be mapped to the elements in the graph DB. The second component constructs the Cypher query by taking as input the information about the intent and the entities extracted by the first component.

### A. Intent Detection and Entity Extraction Component

Given a plain text question, the first step is to detect the question intent (i.e. what should be returned by the corresponding query) and to extract the main entities that correspond to elements in our graph database. Intent detection and entities extraction fall under natural language processing (NLP) and understanding (NLU) for which state of the art techniques can be applied. In our work, we use Wit.ai[5] as a framework to perform these tasks. In the following, we define the notions of *intent* and *entity recognition* in the context of event data related questions and we explain how Wit.ai is configured accordingly.

*1) Intent Definition:* An intent describes the question purpose and assists the system in determining what type of information should be returned, and consequently which minimal graph elements are required to construct the Cypher query. In general, process queries can be summarized into three main categories according to the type of information they return:

- *structural queries*: these queries answer questions about event related data. In terms of our event property graph, the queries can return (i) nodes and/or their data attributes

[5]https://wit.ai/

and (ii) aggregated information by applying aggregate functions (e.g. count, maximum, etc.). Examples of questions to this category are shown in Table II- column2.
- *behavioral queries*: these queries answer questions about the temporal order between activities. They focus on the directly follow relations between activity instances.
- *performance queries*: These queries return performance-related data, such as the duration or time between specific activities. These information are not directly encoded in the graph data model but are computed based on stored timestamp data attributes.

For each of the above categories, a set of intents should be defined. The intent allows us to determine the *return clause* and the form of the *match clause* in the constructed Cypher query. In the current work, we focus on the intent definition and Cypher query construction of *structural queries*. The behavioral and performance queries are left for future works since they require dedicated intent definition and Cypher query construction mechanisms.

First, we define a set of generic intent patterns that are applicable to any data stored in a labeled property graph. Then, we show how these patterns can be instantiated according to our event property graph meta-model shown in Fig.1. The set of intent patterns and their instantiations are shown in Table I. In the following, we explain each of these patterns.

***Node pattern (P1)***: This intent corresponds to questions that request an information about an individual node type without taking into account any relationship. For example, in the question *'Which activities were executed today?'*, the user inquires about the activity node types whose execution date property has as value the current date. The possible instantiations of this pattern are *Role, Actor, Activity* and *Artifact*. Examples of questions for the *Activity* and *Artifact* intents are shown in Table II.

***Node_Relation pattern (P2)***: This intent is linked to questions that request the information about a node type while taking into account the relationship that exists between this node and other node types in the graph. For example, in the question *'Who was involved in processing Application_2110141037?'*, the user inquires about the actor who processed a specific application. To find such information, the query should match the activity node that corresponds to processing a specific application and return the actors that are connected to this node through the relation *HasContribution*. The instantiations of this pattern are all possible pairs of (node type, relation type) that exist in the graph in Fig.1.[6] It is worth noting that we add all possible pairs regardless of the relation direction. For instance, the intents *Role_ActorWithRole* and *Actor_ActorWithRole* are two possible intents. The former is linked to questions about the role of specific actors while the latter is linked to questions about the actors having specific roles. Examples of questions for the *Actor_hasContributed* and *Artifact_AffectArtifact* intents are shown in Table II.

[6]We exclude the DirectlyFollow and A_DirectlyFollow relations since we consider only structural queries as already explained

***Node_AggFunc (P3)*** and ***Node_Relation_AggFunc (P4)*** patterns are similar to *Node* and *Node_Relation* patterns respectively, but instead of returning nodes, they return the result of one of the aggregation functions: COUNT, MAX, MIN, AVG or SUM. Examples of questions are shown in Table II.

A machine learning based approach is followed for intent detection. Wit is trained with a set of questions labeled with potential intent instantiations. Table III shows an example of a question and the intent detected by Wit.

*2) Named Entity Recognition:* Entities are elements of interest extracted from a plain text question and used to build the Cypher query. Entity extraction is a commonly performed task in NLP and is known as Named Entity Recognition (NER). To perform NER, a list of predefined categories with which extracted entities will be tagged must be created. In our work, entity tags refer to elements in the event property graph which can be *node and relation labels, their properties and values*. Therefore, we employ a NER that tags entities extracted from the input plain text question with one of the node labels (e.g. activity, actor, artifact, etc.), edge labels (e.g. HasContributed, AffectArtifact, etc.), node/edge properties or property values. Formally, a tag is defined as follow:

**Definition 2 (Tag Set).** *Let $G_E = (N, E, \gamma, \lambda, \rho, \sigma)$ be an event property graph, $L_N$ be the set of nodes' labels and $L_R$ bet the set of relations' labels. The set of tags to which an entity extracted from a plain text question can be mapped is defined as $T = L_N \cup L_R \cup P_G \cup V_G$ where:*
- *$P_G = \{(\lambda(x), \rho(x)) \mid x \in N \cup R\}$; is the set of tags that correspond to nodes' and edges' properties;*
- *$V_G = \{(\lambda(x), \rho(x))_v \mid x \in (N \cup R) \times P\}$ is the set of tags that correspond to properties' values.*

**Definition 3 (Entity-Tag Set).** *Let $T = L_N \cup L_R \cup P_G \cup V_G$ be the tag set, and $E$ be the set of extracted entities from an input plain text. The output of the NER step denoted as $ET = \{(e, t) \mid e \in E \ \wedge \ t \in T\}$ is the set of all extracted entities associated with their corresponding tags. We denote by $T_E$, the set of tags present in $ET$.*

To perform NER in Wit, a Wit entity is created for each possible tag in $T$ (as given in Definition 2). Tags for numerical and date values are excluded since Wit can automatically detect them with the help of the wit/number and wit/date tags. Since each of the created Wit entities may appear in several forms in the question, Wit allows to add a set of synonyms for each entity. For instance, the *Actor* entity could appear in the question as: actor, user, resource, person. It is important to note that the accuracy of entities extraction increases as the number of added synonyms increases. Table III shows the entity-tag set extracted for the given question.

### *B. Query Construction Component*

The query construction component takes the detected intent and the extracted entity-tag set from the previous step and constructs a Cypher query. The detected intent defines the minimal required elements that should appear in the Cypher

| | Intent Pattern | Intent Instantiation |
|---|---|---|
| P1 | Node | Role, Actor, Activity, ArtifactType |
| P2 | Node_Relation | Role_WithRole, Actor_WithRole, Actor_HasContributed, Activity_HasContributed, Activity_AffectArtifact, ArtifactType_AffectArtifact, etc. |
| P3 | Node_AggFunc(count/Max/Min/Sum/Avg) | Role_AggFunc, Actor_AggFunc, Activity_AggFunc, ArtifactType_AggFunct |
| P4 | Node_Relation_AggFunc | Role_WithRole_AggFunc, Actor_WithRole_AggFunc, Actor_HasContributed_AggFunc, Activity_HasContributed_AggFunc, Activity_AffectArtifact_AggFunc, etc. |

TABLE I: Intent patterns for structural queries and their instantiations for event property graphs

| Intent | Example questions |
|---|---|
| Activity **(P1)** | Which activities were executed today? |
| Application **(P1)** | Which applications have requested amount more than 15000? |
| Actor_HasContributed **(P2)** | Who was involved in processing Application_2110141037? |
| Application_AffectArtifact **(P2)** | Which application with type New credit was incomplete? |
| Activity_count **(P3)** | How many activities were executed last month? |
| Application_Max **(P3)** | What is the maximum requested amount for those applications with loan goal Home improvement? |
| Offer_CancelledOffer_Max **(P4)** | What is the highest number of offers canceled in a single application? |
| Activity_HasContribution_count **(P4)** | What is the total number of UserAG contributions in Application_1765444083? |

TABLE II: Examples of questions related to the loan application process with their corresponding intents

| | |
|---|---|
| Question | Return all car loan applications validated within the last three months, and which received 3 offers minimum, order them by their requested amounts. |
| Detected Intent | Application_AffectArtifact (**P2**) |
| NER (entity, tag) | (car, (Application, LoanGoal)$^v$), (applications, Application), (applications validated, (Activity, name)$^v$), (last three months, wit/date), (3, wit/number), (offers, Offer), (requested amounts, (Application, requestedAmount))$^{\cdot}$ |
| Cypher Query | **MATCH** (application: Application)-[:Offered]-(offer: Offer), (activity: Activity)-[:AffectArtifact]-(offer), (activity)-[:AffectArtifact]-(application)<br>**WHERE** application.LoanGoal= 'Car' AND activity.Name= 'A_Validation' AND activity.completeTime>='2021-03-01'<br>**WITH** application, COUNT(offer) as offerCT<br>**WHERE** offerCT>=3<br>**RETURN** (application)<br>**ORDER BY** application.RequestedAmount |

TABLE III: An example of a question with detected intent and extracted entities, as well as the corresponding cypher query.

query. In case some of these elements are not extracted, our approach infers them automatically. On the other hand, any additional extracted entity-tag element enriches the query with additional constraints and/or matched sub-graph elements.

Let $I$ be the detected intent and $ET$ be the extracted entity-tag set. $T_E$ is the set of tags in $ET$. The patterns of the five main clauses (MATCH-WHERE-WITH-ORDER BY-RETURN) of the query are constructed as detailed below.

`MATCH clause`: The MATCH clause consists of either a single node or a matched sub-graph. In case $I$ is an instance of the *Node* or *Node_AggFunc* patterns, 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 II is *Activity*. Therefore, `MATCH (activity:Activity)` is constructed.

Otherwise, a sub-graph should be added to the clause. The sub-graph is automatically identified out of $I$ and $E_T$ and should be conform to the event property graph meta-model illustrated in Fig. 1. To do so, all node and relation types that appear in $I$ and $E_T$ are added, including the node and relation types that are extracted from the property and value tags (see $P_G$ and $V_G$ in Definition 2). Then, missing nodes/relations are inferred to form a connected subgraph and are added. For example, the intent of the question in Table III is *Application_AffectArtifact*. Therefore the node *Application* and the relation *AffectArtifact* are added. From the extracted entity-tag set, we add the nodes *Activity* and *Offer*. Afterwards, the missing nodes/relations are added as follow: The relation *AffectArtifact* is added to connect i) *Activity* and *Application*, and ii) *Activity* and *Offer*. The relation *Offered* that connects the nodes *Application* and *Offer* is also added. The MATCH clause becomes as shown in the table.

`WHERE clause construction`: The WHERE clause is made up of a conjunction of triples[7] (property, operator, value) where property and value appear in $E_T$. Values can be of textual, numerical or date type. Textual values are tagged with a value tag from which their associated property is extracted and the "=" operator is added. In our example in Table III, in the extracted entity-tag $(car, (Application,\ LoanGoal))^v$: *car* is an extracted entity labeled with the value tag $(Application,\ LoanGoal)$ where *LoanGoal* is a property of the node type *Application*. Therefore, the triple `application.LoanGoal = 'car'` is added.

Date tags extracted as wit/date are automatically associated to the timestamp property of an activity node type. Wit extracts a date as either a single date, for which the "=" operator is added, or an interval, for which two conjunctions of triples are created with "$\leq$" and "$\geq$" operators respectively.

Numerical values are extracted under wit/number tag and

[7]The current version supports the conjunction of conditions

should be assigned to their corresponding properties with the correct 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 $> 15000$. Existing works in NLI systems make assumptions about the order of appearance of the triple (operator, property, value) in the question. For example, they assume that the numerical value always comes directly after the operator name. We do not impose such constraints in our work. Instead we propose to find the valid (property, operator, value) by computing a distance based on words closeness to support all possible questions syntax.

To do so, we define trigger words to extract the operators from the question (e.g. 'at least', 'minimal' in the question are trigger words for the operator '$\geq$'). Out of the extracted operators, numerical values and properties, the system tries all possible triples (property, operator, value) combinations, and selects as valid triples those whose elements are close to each other in the question according to a user specific threshold. The closeness is computed based on the distance between the elements of each pair in the triple and is defined as the average distance between each pair of elements.

$$\begin{aligned} closeness(e_1, e_2, e_3) &= \frac{\sum_{i,j \in [1,3]} (dist(e_i, e_j))}{3} \\ dist(e_i, e_j) &= \sum_{k \geq 0} penalty(e_k) \end{aligned} \tag{1}$$

where $e_1,\ e_2,\ e_3$ are the elements of the triple. The distance $dist$ between two elements $e_i$ and $e_j$ is the sum of penalties assigned to the entities $e_k$ separating them in the question. A penalty of 0, 1 or 2 is assigned to each entity in the question based on its part of speech tag that influences the semantic closeness of the pair of words. Verbs are assigned the highest penalty 2 since they should not occur between the elements of the triple; nouns are assigned the penalty 1 and all other tags are assigned the penalty 0.

`WITH clause construction`: The WITH clause is constructed after the WHERE clause if there is a need to an aggregation. We treat the presence of trigger words (e.g. in each, for each, by, etc.) that may indicate the need of an aggregation based on the closest entity following it. The proposed algorithm uses regular expressions to find the closest entity to the trigger word. Furthermore, this clause defines the variable of the count function if there is a condition on it. For example in the example in Table III, 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 condition that it should be greater than 3. The system handles conditions on count variables in the same way that it handles conditions on property values. It tries to identifies the valid triples (operator, node, number) where *node* is the entity extracted from the plain text associated with a node tag in $T_N$.

`ORDER BY clause construction`: The ORDER BY clause is added if we detect trigger words (e.g., ordered, descending, sorted, etc.) in the question. In this case, a regular expression determines the closest extracted entity that follows the trigger word. The entity tag can be either a property or a node label. If it is a property, an order by on the property is added. Otherwise, an order by on the count of the node label is added. For example in Table III, the trigger word is *order* and the closest entity is the property requested amount.

`RETURN clause`: Two types of information could be returned: (i) If the intent is an instance of the intent patterns *Node* or *Node_Relation*, the node type that appears in the first part of the intent is returned (see example in Table III). In some cases, when a user inquires about only some properties of this node, the system returns those properties rather than the entire node. Such case can be recognized by examining the property names in $E_T$ that do not have any corresponding value tag. (ii) If the intent is an instance of the intent patterns *Node_AggFunc* or *Node_Relation_AggFunct*, the result of an aggregation function is returned. Similarly, the node type that appears in the first part of the intent determines to which node the aggregation function should be applied.

## V. Evaluation

The approach has been implemented as a standalone python application with a conversational interface. The application connects to Wit.ai, where an application is configured to extract graph entities, and to Neo4j, where the event property graph is stored. The source code and a detailed description of the application configuration can be found at http://www-inf.it-sudparis.eu/SIMBAD/tools/ProcessChatBot.

To evaluate our approach, we used the BPIC'17 event log [10] which contains data describing a loan application process from filling out a loan application to decision-making (approving or declining). The log has been filtered by removing infrequent events whose frequency is less than 20%. As a result, we obtained 2105 unique event records for 26 different activities. There are 154 applications, 183 offers, and 240 workflows. The activities were performed by 69 actors, and no business roles were included. The data is stored as a graph database in Neo4j, yielding 2157 nodes and 10692 relations.

We performed two main experiments to assess the feasibility and to evaluate various aspects of the approach in a controlled environment. In the first experiment, we evaluated the performance of the intent detection and entities' extraction component (Section V-A). While in the second experiment, we evaluated the query construction's accuracy taking into account the performance information collected from the first experiment (Section V-B). In Section V-C, we discuss the threats to the validity of the performed experiments. All details related to the data used and the results obtained can be found at http://www-inf.it-sudparis.eu/SIMBAD/tools/ProcessNLI.

### A. NLP tasks evaluation

Since the query construction component largely depends on the correct detection of intents and entities, we evaluated in

| | Min | Max | Avg |
|---|---|---|---|
| Precision | 0.66 | 1 | 0.9807 ($\pm$0.0679) |
| Recall | 0.6 | 1 | 0.9699 ($\pm$0.0994) |
| F-Measure | 0.66 | 1 | 0.9772 ($\pm$0.0783) |

TABLE IV: Evaluation metrics

this experiment the performance of Wit.ai in intent detection and entities' extraction.

*1) Experimental setup:* A Wit.ai application is created and a script is written to automatically generate the Wit entities from the set of possible tags (see Definition 2). For each Wit entity, a set of synonyms is manually added.

As for the intent detection part, Wit is trained with a set of questions labeled with the corresponding intent as defined in Table I. For each possible intent, we created 10 questions which are linguistically similar but written in different syntax. The trained application was then tested with 100 questions that were not used in the training phase.

To assess the performance of Wit, we computed the accuracy of intent detection, and the precision/recall/F-score of the entities extraction. The accuracy is computed as the number of questions with correctly detected intent divided by the total number of questions. For each question, the precision/recall/F-score of entities extraction is computed. The precision for a given question 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 questions.

*2) Results:* An accuracy of 0.94 is obtained for the intent detection part which indicates that Wit intent detection is robust against linguistic variation. As any machine learning based approach, the more training data we have, the better accuracy results we may obtain.

Table IV shows the results of the entities' extraction part. In average, an F-score of 0.977 is obtained. The results indicate that, overall, Wit is able to distinguish between graph entities, and to recognize nodes, relationships, attributes and their values that occur in different syntax in the question. By closely inspecting the results, we are able to underline the following observations at the level of entity extraction.

First, we analyzed the low precision values which indicate that some entities were incorrectly tagged. This was mainly due to insufficient information in the question and to Wit entities that may have common keywords/synonyms. For example, given the question *'What is the loan goal of application with amount 7500?'*, and the extracted entity *amount*. There are two entities in Wit with which *amount* can be tagged: 1) the attribute *Requested Amount* of the artifact node *Application* or 2) the attribute *Offered Amount* of the artifact node *Offer*. By looking at the question context, it becomes trivial that the extracted entity should be tagged with the first option. However, Wit does not take into account the sentence context, and as a result, assigns a tag randomly.

In addition, we found that many of the incorrectly extracted entities are due to insufficient number of synonyms added to Wit entities, especially for activity names. The same applies for low recall values in which Wit missed some entities that had to be extracted due to insufficient synonyms. For example, in the question *'Give me the offers in applications with a budget of 25000 or less'*, Wit was unable to detect that 'budget' refers to the attribute *offered amount*.

To summarize, the performance of this step can be increased by simply increasing the number of synonyms added to Wit. However, the problem of context-aware NER remains an open challenge that we aim at addressing in future works.

### B. Query construction evaluation

In this experiment, we evaluate the query construction component. By design, our constructed queries are syntactically correct. Therefore, we evaluate whether they are semantically correct (i.e. they return the correct result as inquired by the user). We compare our intent-based approach to a baseline that does not involve an intent detection step.

*1) Experimental setup:* We designed a set of questions for which the detected intent and the extracted entities of the NLP component are correct. The questions are grouped into two categories according to their intent: i) questions that fall under the intent patterns *Node (P1)* or *Node_AggFunc (P3)* and ii) questions that fall under the intent patterns *Node_Relation (P2)* or *Node_Relation_AggFunc (P4)*. For the second category which requires to construct a subgraph in the match clause, the questions are designed in decreasing level of completeness. Some questions explicitly include all the necessary details and graph elements and do not require any detection of implicit node/relation to construct the subgraph while others require the detection of implicit nodes/relations.
We compute the accuracy by dividing the number of semantically correct queries by the total number of questions.

*2) Results:* Fig. 3 shows the accuracy of the constructed queries with and without intent information. Both systems achieve, in average, a similar accuracy of 0.95 in the first category of questions (P1, P3). This is explained by the fact that these questions are simple and require only one node type to be matched and to be returned.

The intent-based system clearly outperforms the baseline in the second category of questions with an average accuracy of 0.94 versus 0.52. In this category, the intent helps the system to match the correct subgraph and to return the correct information. In the baseline, the system tries to construct a subgraph out of the extracted entities without taking into account the question context. To return the result, we equipped the baseline system with trigger words. For example, to indicate that the return clause should contain the aggregate function *count*, we add some trigger words such as 'How many' and 'What is the number of'. However, keyword matching is limited and cannot account for the linguistic variation.

### C. Threats to Validity

The first threat to validity is related to the evaluation with a small set of questions as opposed to a publicly available bench-

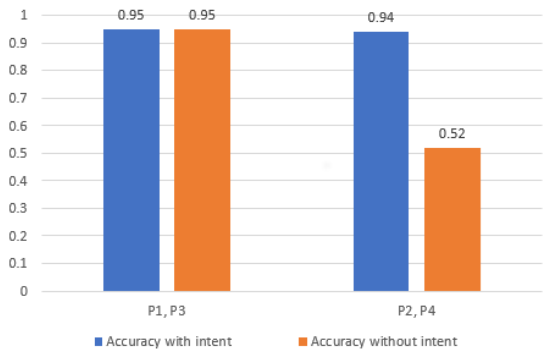


Fig. 3: The accuracy of queries with and without intent

mark. The objective of the performed experiments is to show that our approach is feasible and to evaluate various aspects of the system in a controlled environment. To generalize the results, we will work on creating a benchmark by collecting questions from experts in the domain.

Another threat is the absence of a comparative analysis with existing works. First, in terms of process querying, existing process querying works have a different objective since their aim is to propose new formal querying languages. In our work, we do not propose a new language, instead, we propose a NLI to translate a question to a structured query. Our approach can complement existing works since it can be implemented as a layer on top of formal languages. Second, in terms of natural language interfaces to databases, existing works propose to construct SQL, SPARQL or Cypher queries. However, they use different data and benchmarks for the evaluation. Our data is process-oriented which makes the techniques and evaluation objectives different.

## VI. Conclusion

We proposed an intent-based NLI for querying process execution data. The interface facilitates the querying activity by understanding and interpreting the intent of the user from a natural language question, constructing automatically the corresponding Cypher query to be executed over the process data stored in a graph database, and returning the answer. We take advantage of both machine and rule-based approaches, and we proposed a hybrid NLI system. We evaluated our approach using a real-life event log from BPIC'17. Experimental results showed that our intent-based outperforms the baseline system with an average accuracy of 0.94 versus 0.52.

Our main focus for the future works is to generalize the results by creating a benchmark of questions collected from experts in the domain and that cover different domains. We also aim to conduct a case study to assess the satisfaction of users with the NLI system as compared to querying tools. In addition, we are working on extending the system with behavioral and performance queries as discussed in the paper.

## References

[1] van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition (2016)
[2] Affolter, K., Stockinger, K., Bernstein, A.: A comparative survey of recent natural language interfaces for databases. VLDB J. **28**(5), 793–819 (2019)
[3] Angles, R.: The property graph database model. In: Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management. CEUR Workshop Proceedings, vol. 2100 (2018)
[4] Beheshti, S., Benatallah, B., Nezhad, H.R.M., Sakr, S.: A query language for analyzing business processes execution. In: Business Process Management - 9th International Conference. vol. 6896, pp. 281–297 (2011)
[5] Beheshti, S., Benatallah, B., Sakr, S., Grigori, D., Motahari-Nezhad, H.R., Barukh, M.C., Gater, A., Ryu, S.H.: Process Analytics - Concepts and Techniques for Querying and Analyzing Process Data (2016)
[6] Bergamaschi, S., Guerra, F., Interlandi, M., Lado, R.T., Velegrakis, Y.: Combining user and database perspective for solving keyword queries over relational databases. Inf. Syst. **55**, 1–19 (2016)
[7] Blunschi, L., Jossen, C., Kossmann, D., Mori, M., Stockinger, K.: SODA: generating SQL for business users. Proc. VLDB Endow. **5**(10), 932–943 (2012)
[8] Dijkman, R.M., Gao, J., Syamsiyah, A., van Dongen, B.F., Grefen, P., ter Hofstede, A.H.M.: Enabling efficient process mining on large data sets: realizing an in-database process mining operator. Distributed Parallel Databases **38**(1), 227–253 (2020)
[9] Dong, L., Lapata, M.: Language to logical form with neural attention. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL (2016)
[10] van Dongen, B.: BPI Challenge 2017 (2017). https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
[11] Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. CoRR **abs/2005.14552** (2020)
[12] de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Connecting databases with process mining: a meta model and toolset. Softw. Syst. Model. **18**(2), 1209–1247 (2019)
[13] Polyvyanyy, A.: Business process querying. In: Encyclopedia of Big Data Technologies (2019)
[14] Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decis. Support Syst. **100**, 41–56 (2017)
[15] Räim, M., Ciccio, C.D., Maggi, F.M., Mecella, M., Mendling, J.: Log-based understanding of business processes through temporal logic query checking. In: On the Move to Meaningful Internet Systems: OTM. vol. 8841, pp. 75–92 (2014)
[16] Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U.F., Mittal, A.R., Özcan, F.: ATHENA: an ontology-driven system for natural language querying over relational data stores. Proc. VLDB Endow. (2016)
[17] Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: Advanced Information Systems Engineering - 28th International Conference, CAiSE. vol. 9694, pp. 290–305 (2016)
[18] Shekarpour, S., Marx, E., Ngomo, A.N., Auer, S.: SINA: semantic interpretation of user queries for question answering on interlinked data. J. Web Semant. **30**, 39–51 (2015)
[19] Sun, C.: A natural language interface for querying graph databases. Master report, Massachusetts Institute of Technology (2018)
[20] Tan, Z., Zhang, J., Huang, X., Chen, G., Wang, S., Sun, M., Luan, H., Liu, Y.: 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. pp. 116–122 (2020)
[21] Yongsiriwit, K., Chan, N.N., Gaaloul, W.: Log-based process fragment querying to support process design. In: 48th Hawaii International Conference on System Sciences, HICSS. pp. 4109–4119 (2015)
[22] Zheng, W., Cheng, H., Zou, L., Yu, J.X., Zhao, K.: 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. pp. 217–226 (2017)
[23] Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR **abs/1709.00103** (2017)