Formal Ontology in Information Systems
T. Prince Sales et al. (Eds.)

doi:10.3233/FAIA250500

Process Representation Meets Operational Realization: An Architecture for Data-Driven Process Ontology Application Through Process Mining

Riley MOHER [a,1], Michael GRUNINGER [a]
[a] *Department of Mechanical and Industrial Engineering, University of Toronto*
ORCiD ID: Riley Moher https://orcid.org/0009-0002-3404-3386, Michael Gruninger https://orcid.org/0000-0003-3925-2398

Abstract. Processes are fundamental to enterprises, serving as significant engines of optimization and analysis. Understanding business processes is critical, yet integrating formal process ontologies with enterprise data and workflows remains difficult. We refer to this specific kind of ontology application, intended for practitioners working with enterprise data, as *operational realization*. The varied ontological commitments and highly expressive representation languages of process ontologies create barriers for operational realization, including issues of decidability, a lack of tooling, and operational constraints. This paper presents an architecture for the operational realization of process ontologies, driven by process mining needs. A key aspect of the architecture is the formalization of ontological commitments in tasks like data cleaning and analysis, which rely on implicit assumptions embedded in the interpretation of process data. Our approach builds on existing methodologies, notably ontology-based data access (OBDA), while going further by encoding domain knowledge required to interpret and reason with process data. This structure moves beyond an A-Box and T-Box distinction, explicitly capturing how the process ontology, domain data, and supporting data interpretation theories enable complex process reasoning. By structuring the dynamics of a process ontology, domain data, process data theories, and by characterizing reasoning scenarios, our approach provides a pragmatic foundation for integrating process ontologies into data-driven process workflows. We demonstrate this architecture with real enterprise data, challenge problems, and scenarios already widely used for benchmarking in process mining.

Keywords. process mining, process ontology, ontology application, ontology-based data access, operational realization

1. Introduction

Processes are diverse constructs, interpreted across many domains, and integral to enterprise decision-making. Their management and analysis drive value, especially in to-

[1]Corresponding Author: Riley Moher, riley.moher@mail.utoronto.ca

day's era of massive, rapidly-streaming data. Process mining has become a key driver of this analysis, offering a data-driven, bottom-up approach to understanding and improving processes. Through the analysis and interpretation of event logs, this practice enables organizations to discover and enhance process models, assess conformance, and predict future behaviours [1, 2]. However, process mining faces significant issues with semantic heterogeneity and data quality in practice, leading to a reliance on ad-hoc and informal methodologies [3, 4].

Enter formal ontology, offering a highly structured and rigorous approach to represent and reason about processes. From domain-specific [5, 6] to broad and expressive upper ontologies [7, 8], these models formalize key process concepts such as activities, events, agents, and time. While these models are ontologically robust and well-formalized, the question of their *operational realization* — their systematic integration with data in practical analysis pipelines — remains unaddressed. Given the reality of noisy and incomplete data, this operational gap becomes a significant barrier, resulting in ontologies becoming merely reference models, subject to ad-hoc interpretations, and undermining their potential strengths.

In this paper, we introduce an architecture to *operationally realize* process ontologies, driven by real applications of process knowledge and data from process mining. Our methodology supports the mapping of process data to ground-level ontology facts, formally facilitates the integration of domain-specific supplementary knowledge, and enables reasoning over practical scenarios. While elements of this architecture have a basis in existing research, including ontology-based data access (OBDA) [9] techniques, our architecture considers the necessary theories to support interpretation and analysis of the data through a component we call the process *data theory*. While OBDA seeks to enable meaningful access to data through ontology-based querying, we go beyond data access and support a wide range of reasoning scenarios drawn from enterprise process mining applications. This more holistic and application-oriented focus ensures that our architecture not only prioritizes the representational strengths of the ontology but also its operational efficacy in dynamic, data-rich environments.

We make the following research contributions: (1) Specification of an architecture for the operational realization of process ontologies. (2) Preliminary validation of the architecture using real-world benchmark datasets, challenge problems, and applications from process mining.

Our architecture is presented in this paper through the following structure: Section 2 first provides expanded background, motivation and key pieces of related work for our approach, including the background on process mining with the application of formal methods, and the importance of operational realization of ontologies. In Section 3, we specify our architecture with major components including the role of the process ontology, the mapping process, the data theory, and the reasoner. Section 4 then provides a validation of our architecture by demonstrating its applicability to real-world datasets and problems. These datasets and challenge problems are drawn from well-known benchmark problems in process mining [4]. Section 5 concludes the paper by summarizing our contributions, discussing the current implementation, its limitations, and an outline for future work, notably employing our architecture for the benchmarking of process ontologies.

2. Process Mining Foundations and Ontology Applications

2.1. Mining for Meaning

Process mining transforms process data into valuable measurements and aggregations of business process knowledge [1,2]. Key tasks within this practice include process discovery, which infers process models from event logs, concept drift analysis, which tracks changes in process execution over time, and predictive process monitoring, which predicts future events and timings for a given process occurrence [1]. These tasks rely on a wide range of process knowledge, yet are all grounded in a common data source: event logs.

Event logs are commonly recorded in the extensible event stream (XES) format [10], which is intentionally flexible, defining an event as "an atomic granule of activity that has been observed." This flexibility enables process mining to be applied across a diverse set of domains but also introduces significant semantic heterogeneity. Events in XES logs can encode many process entities, including status values, object states, or occurrences. Interpreting this data requires domain knowledge to bring out the intended meaning and structure of the data. Additionally, the quality and granularity of event data vary across datasets, making the integration of event logs more challenging. For instance, within widely used benchmarks like the Business Process Intelligence Challenges (BPIC), activity labels can range from a handful to over six hundred [1].

The lifecycle representation of events further complicates interpretation. The XES lifecycle extension defines standard event transitions such as "start," "complete," "schedule," and "abort." However, many real-world logs omit lifecycle tags, or use non-standard labels, such as in BPIC 2013, where activity labels like "Accepted" and "Queued" are refined with lifecycle transitions such as "Awaiting Assignment." Without ontological grounding or standardization, this variability makes it difficult to reconcile data across different datasets and domains. While efforts like object-centric event logs (OCEL) [11] attempt to alleviate some of these issues by better structuring logs around objects like orders, shipments, or diagnoses, semantic heterogeneity of events and informality in data handling remain critical, unaddressed issues.

Despite these challenges, process mining practitioners consistently extract meaningful insights by interpreting event data through domain-specific process knowledge (albeit in an informal and ad-hoc manner) [4]. For instance, BPIC 2012 includes a challenge to "identify which decisions have greater influence on the process flow." Addressing such a task requires reasoning about decision-making, control flow, and causality—concepts not explicitly defined in event logs. This critical relationship between the analysis and interpretation of event logs together with a process ontology is formalized as a core part of our architecture, further explained in Section 3.

While these challenges remain, there have been attempts to apply formal methods in process mining, though they lack an *ontology-aware* foundation.

Event Knowledge Graphs (EKGs) [12] extend process mining by representing process perspectives based on objects (e.g., purchase orders or shipments) using labelled property graphs. These methods, however, rely on implicit mappings and lack transparency, limiting their reuse. More recent refinements [13] introduce "semantic headers"

[1] We provide our own quantitative analysis of these datasets available at purl.org/ontologydrivenprocessanalysis/fois25

in JSON format, but these weak and informal encodings are problematic for verification. Similarly, the DECLARE framework [14] models process constraints using linear temporal logic (LTL) but has ontological limitations. It fails to explicitly address semantic heterogeneity and overlooks important process relationships, such as event-state and participation relationships. Other extensions to DECLARE [15, 16] extend this approach by incorporating additional attributes and possible constraints, but still reduce rich process concepts to simple attribute-value pairs. These methods illustrate the existing disconnect between the ontological foundations of process mining and the complex reality of data integration in practice.

2.2. Operational Realization: Application Beyond OBDA

Operational realization refers to the practical and grounded application of an ontology within real-world data-driven systems, bridging the gap between ontological representation and use in everyday enterprise operations. While ontologies typically aim to formalize a domain's conceptual structure, their application—especially in dynamic environments like process mining—requires more than a conceptual model. The challenge of operational realization is in applying an ontology in a way that allows it to be operationally useful, necessarily addressing issues of data quality, incompleteness, flexibility, and accommodating domain-specific interpretations. Existing approaches including ontology-based data access (OBDA) [9] focus on mapping data to an ontology and ensuring meaningful access to the data. Operational realization, however, goes beyond just data access; it must also support a range of reasoning tasks and integrate tacit knowledge that is necessary for interpretation of the data. Thus, in addition to mapping data to a formal ontology, the approach must provide mechanisms to handle the complexity and messiness of real-world data, including the need for flexible and domain-specific interpretations that cannot be captured through a priori mappings.

OBDA, as exemplified in process mining by tools like Onprom [17], maps data to RDF graphs to enable SPARQL queries for entity-centric retrieval. While this provides a useful interface for accessing data via a conceptual model, OBDA is limited to the data and the ontology it maps to. It lacks the means to capture additional knowledge or assumptions—such as effects and preconditions, or the disambiguation of events—that are essential when dealing with real-world data.

For instance, a hospital log with multiple consecutive `Triage Assessment` events may, in reality, represent distinct actions, like an initial assessment and a follow-up [3]. Resolving such issues requires domain-specific assumptions and interpretative cues, which are not captured by the raw data or the process ontology. Issues such as these should be handled in a controlled manner, which requires a means of explicitly capturing the kinds of knowledge being applied to identify and correct these issues. Therefore, operational realization must go beyond OBDA by explicitly integrating this contextual and interpretative knowledge to enable more than just data access, but data understanding.

3. Methodology: Assembling an Architecture

In this section we will provide a specification of the essential components of our architecture. A high-level overview of the architecture is depicted in figure 1.

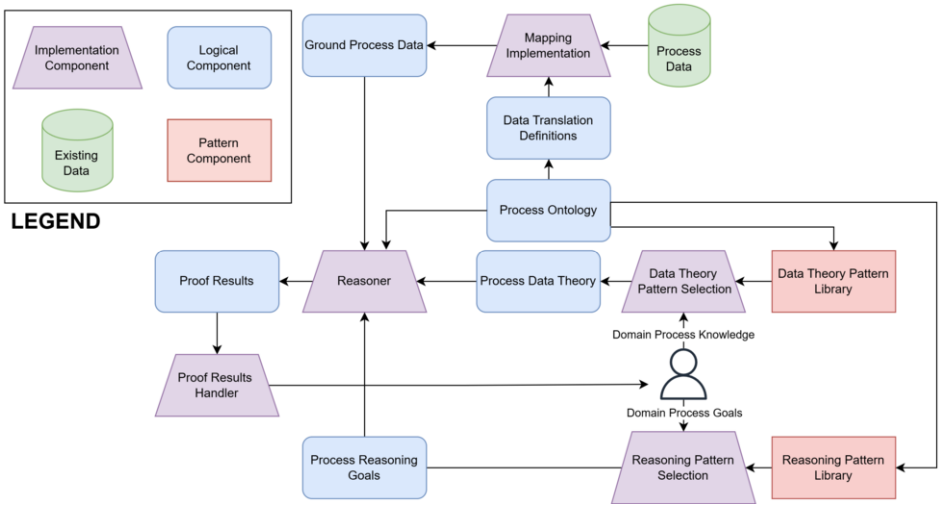

Figure 1. System Architecture for Operational Realization

The architecture consists of many components which interact with one another to produce a tool designed for process mining practitioners as the end-users. The components can be classified into logical, implementation, data, and pattern components, as depicted through the distinct shapes and colours present in figure 1. The implementation components are pieces of software, and serve to integrate the logical components and data, and perform reasoning. The logical components are the formal knowledge artifacts present in the architecture, including both a-priori components like the process ontology, and mapped or generated components including the ground process data, process data theory and any proof results obtained. The pattern components are meta-logical specifications of process knowledge, similarly to how languages like DECLARE [14] or modelling languages provide a vocabulary of constraints or control flow constructs, we provide a library of patterns to be instantiated to specify process constraints and queries in the language of the process ontology. Our approach to knowledge patterns is presented in greater detail as the focal point of [18]. Arrows in the diagram represent both tangible input of data from one component to another as well as referential links, as in the case of the process ontology's pattern libraries.

While the architecture specification does not mandate a specific implementation language, the demonstrative validation presented in Section 4 is implemented using Python (for its use in general-purpose data analysis), illustrating one viable approach.

The Process Ontology is the core of the architecture—every other component depends on it either directly or through an intermediary component. The process ontology serves as a foundational formalization of processes, providing an axiomitization of process concepts and relationships that is domain-independent. This approach is ontology-agnostic and supports an arbitrary ontology, expressed in various languages and levels of expressiveness. For validation and demonstration we use the Process Specification Language (PSL) [19] in Section 4. Because the ontology is the central element of the architecture, the choice of ontology necessarily has consequences for the implementation choices of

other components. These choices and the impact of an ontology will be discussed as the architecture is explained here.

The Mapping is specified by the triple $\langle \mathscr{T}_{\mathscr{P}}, \Gamma, \mathscr{S} \rangle$, where $\mathscr{T}_{\mathscr{P}}$ is the process ontology, Γ is the data translation function, and $\mathscr{S}$ is the log schema. The data translation function implements a set of mappings of the form $q(\mathbf{x}) \implies \phi(\mathbf{x}')$, where $q(\mathbf{x})$ is a query template over $\mathscr{S}$ and datum $\mathbf{x}$, and $\phi(\mathbf{x}')$ is a ground sentence in the language of $\mathscr{T}_{\mathscr{P}}$, including the constants $\mathbf{x}'$. In practice, the form of Γ depends on the expressiveness of $\mathscr{T}_{\mathscr{P}}$. For traditional OBDA, Γ is specified using RML, based on the OWL encoding of the process ontology, with $\phi(\mathbf{x}')$ taking the form of either class assertions (`x rdf:type C`) or property assertions (`x1 P x2`) [20]. For more expressive process ontologies, the mapping function is more flexible. Here, we describe two main methods: the first extending existing OBDA mapping tools, where Γ is specified with a lightweight RML specification Γ_0 and an additional specification in the language of $\mathscr{T}_{\mathscr{P}}$, Γ_1. The combined mapping function $\Gamma := \Gamma_1 \circ \Gamma_0$ implements a set of mappings composed of two stages: $q(\mathbf{x}) \overset{\Gamma_0}{\implies} \phi(\mathbf{x}') \overset{\Gamma_1}{\implies} \theta(\mathbf{x}')$, where the first stage maps queries q to RDF triples ϕ, and the second maps these triples to process ontology ground terms θ. Notably, the RDF mapping's vocabulary is treated as a reference taxonomy, used purely for implementation convenience due to the mature tooling for RML mapping. No reasoning occurs at the RDF level; it merely serves to facilitate the translation to the more expressive process ontology. The second approach involves directly mapping the log data into the language of $\mathscr{T}_{\mathscr{P}}$. Due to a lack of tooling for such generalizable, flexible, and expressive mappings, we do not focus on the implementation of this method in the present work, but acknowledge it as a potential avenue for future work.

In the context of OBDA, there are two general methods of data access: materialization, where data is transformed into a repository of the mapped logic, and virtualization, where the data remains in its original form and is queried in real-time via the ontology query language of choice (often SPARQL in practice) [20]. While virtualization is well-understood and well-supported for the SQL/SPARQL query dynamic, this is not the case for more expressive languages like modal logics or first-order logic, and for flexible data formats like XES [10]. Because of this lack of available methodology as well as our need to support data theories for ontology applications beyond access of the data, we adopt a materialization approach.

The Process Data Theory formalizes the assumptions and additional process knowledge necessary to correctly interpret and reason with process data. While the process ontology defines the foundational concepts and relationships of processes, and the ground data provides a record of process observations, additional domain-specific constraints and scoping choices influence how recorded events should be understood. Information systems may fail to log events correctly and consistently, or the needs of process analysis may simply require additional contextual information. Consequently, the alignment between recorded data and a practitioner's analysis goals is non-trivial. The process data theory provides a principled way to formalize implicit assumptions, ensuring that reasoning over process data aligns with the intended domain semantics.

To represent this knowledge, the process data theory is expressed in the language of the ontology via knowledge patterns. These patterns are conservative extensions of the ontology, maintaining consistency while enabling flexible specification of additional constraints. Consider an exemplary instance of a data theory sentence: "When a frag-

ile object is dropped, it breaks." - this can be expressed with the following first-order sentence in the language of the ontology:

$$(\forall o)\, occurrence\_of(o, drop) \wedge prior(fragile, o) \implies holds(broken, o)$$

This sentence is a specific example of a **state-based effect (SBE)** pattern, since it specifies how a state effects the result of an activity occurring. The pattern can be expressed as:

$$SBE(a, f_1, f_2) := (\forall o)\, occurrence\_of(o, a) \wedge prior(f_1, o) \implies holds(f_2, o)$$

In natural language, this states: "If an action a occurs and a state f_1 holds prior, then another state f_2 must hold afterwards."

To facilitate the use of process data theory without requiring expertise in logical formalisms, we introduce a data theory pattern library. This library is specified at a meta-logical level, defining commonly used logical sentence templates for data theorems. Rather than requiring practitioners to manually encode assumptions in a formal logic, the library contains predefined pattern such as temporal constraints, or activity hierarchies. More details on this pattern approach is the focus of previous work [18].

The Reasoner The reasoner serves as the computational component responsible for logical inference and consistency checking within the framework. Its role extends beyond merely executing reasoning tasks—it also provides structured methods for expressing domain-relevant queries and constraints in a formalized manner. For first-order-logic encoded ontologies for example, this can take the form of an automated reasoner with wrapper software to call queries and verification checks from the same interface data is accessed.

The kinds of reasoning in our architecture can be understood based on the permutations of logical artifacts used, leading to three configurations:

- **Process Ontology + Mapped Data:** Inference and consistency checking over structured event data against foundational process knowledge. Checks if the data matches the "universal process truths"
- **Data Theory + Mapped Data:** Inference over domain-specific process constraints and relationships that do not rely on the process ontology.
- **Process Ontology + Data Theory + Mapped Data:** Reasoning that incorporates both the process data's interpretation as well as foundational process knowledge. The reasoning task is complex enough to go past matching patterns in the data and necessarily references the process ontology.

These configurations allow for flexible reasoning, depending on the kinds of knowledge required for a given task. Additionally, the reasoner's usage can be classified into two categories:

- **Consistency Checking:** Ensuring that the combined knowledge is logically consistent. This can indicate inconsistencies between the foundational world view (process ontology), the domain (data theory), or the data (mapped data).

- **Queries and Rule Verifications:** Provides proofs for a logical goal. Depending on whether a given logical goal is existentially or universally quantified, the proofs are interpreted as query results or rule verification items, respectively.

While it may be desirable both for formality and traceability for reasoning to be done solely via a theorem prover, tractability may necessitate supplementary implementation efforts. For some reasoning tasks, they may not align neatly with the ontology's expressiveness, or in the opposite direction, the logical formulation may be undecidable or intractable for large datasets. Thought must be given to balancing expressiveness and tractability: if the ontology is too limited, ad-hoc methods undermine verifiability and transparency, while high complexity risks undecidability and poor scalability for large datasets.

Similar to the data theory, the architecture includes a reasoning library designed to classify common reasoning tasks into broad, reusable patterns. While this paper does not focus on identifying these patterns, their inclusion supports the practical use of the architecture by leveraging shared process mining tasks across domains and datasets. This allows practitioners to apply structured inference without manually encoding logical goals, improving usability while maintaining expressiveness. Additionally, by drawing from a predefined set of templated logical statements, the library provides control over reasoning complexity, ensuring that queries remain well-formed and computationally feasible.

4. Validation: Architecture in Action

In this section, we validate the feasibility of our approach through demonstrative examples of the architecture's application to real enterprise data and scenarios based on the widely-recognized BPIC challenges [4, 21]. These scenarios consist of logical reasoning problems with reference to our proof-of-concept implementation [1] where relevant. The implementation consists of a process ontology closely based on PSL [19], mapping implemented with python using RML and YARRML [22] (for the sake of human readability) data translation definitions, and a reasoning suite based on Prover9 [2]. Though we include first-order-logic and prover9 encodings for this demonstration, our python mapping implementation supports mapping to RDF, prover9, CLIF, and prolog formats. Our in-progress implementation demonstrates the feasibility of our architecture and is openly available.

To ground the demonstration in meaningful benchmarks, we present process reasoning scenarios that are not only conceptually rich from a process ontology perspective, but practically grounded in real datasets and enterprise needs. These structured challenges require sophisticated process reasoning to address issues including temporal constraints, supply chain compliance, and organizational dynamics. For each example, we provide a description of the problem including a sample of the relevant data used, the domain knowledge required, and a description of the reasoning problem. [3] Additionally, we specify the formal representation including the referenced mapped sentence(s), data theory

[1] https://purl.org/ontologydrivenprocessanalysis/fois25

[2] https://github.com/ai4reason/Prover9

[3] The sample of the data will not be exactly as depicted in the raw dataset purely in the interest of brevity, but the fundamental logical relationships will be maintained. This includes abbreviated timestamp formats, shortened column names, etc.

sentence(s), process ontology axioms, and any goals to be proven. The three examples presented here demonstrate each of the previously mentioned reasoning scenarios requiring, in addition to the mapped data, the process ontology alone, the data theory alone, and both the data theory and process ontology.

4.1. Activity Boundary Validation

This scenario is taken from a dutch financial institution, where the log depicts a loan application process [23]. This scenario is a part of preliminary data validation. The data and representation are in tables 1 and 2.

Table 1. Activity boundary data

Activity	**Transition**	**Timestamp**
Follow-up quote	complete	12:00
Follow-up quote	start	12:02

Table 2. Activity boundary example

Domain Knowledge	None
Reasoning Problem	Detection of inconsistent timestamps (consistency check).
Mapped Sentence	$(\text{activity\_occurrence}(o_0) \wedge \text{begin\_of}(o_0, \texttt{12:02})$ $\wedge \text{end\_of}(o_0, \texttt{12:00}) \wedge \text{before}(\texttt{12:00}, \texttt{12:02}))$
Data Theory	None
Process Ontology	$(\forall x \, \text{activity\_occurrence}(x) \implies \exists t_1, t_2 \, (\text{begin\_of}(x, t_1)$ $\wedge \text{end\_of}(x, t_2) \wedge \text{beforeEq}(t_1, t_2)))$
Goal	None

In this scenario, a simple data quality check is being performed. This is one of many possible simple data quality checks that could be verified for the data that does not require the data theory to be utilized. Here, an inconsistency arises between the mapped data's timestamps and the expected ordering of event timestamps: activities cannot complete before they have started. As part of the validation of the log files, other incorrect data can be identified requiring domain knowledge to be specified as part of the data theory, as in the next example.

4.2. Supply-Chain Compliance

This scenario is taken from an anonymized purchase order system log, where the data tracks purchase orders and their related items [24]. The scenario focuses on ensuring compliance of the process with set requirements. The data and representation are in tables 3 and 4

Table 3. Supply chain compliance data

Activity	**PO**	**Item**
Record Goods Receipt	A	B
Record Invoice Reciept	A	B

Table 4. Supply chain compliance example

Domain Knowledge	Items with recorded goods receipts must have an accompanying recorded invoice receipt.
Reasoning Problem	Ensuring that every recorded good receipt for an item has an accompanying recorded invoice receipt.
Mapped Sentence	$(\text{item}(B) \wedge \text{participates}(B, o_1)) \wedge \text{participates}(B, o_2)) \wedge$ $\text{occurrence\_of}(o_1, \text{Record\_IR})) \wedge \text{occurrence\_of}(o_2, \text{Record\_GR}))$
Data Theory	$(\forall i\, \text{GR\_IR\_Rule}(i) \equiv$ $(\forall o_1\, \text{occurrence\_of}(o_1, \text{Record\_GR}) \wedge \text{participates}(i, o_1)$ $\supset \exists o_2\, \text{occurrence\_of}(o_2, \text{Record\_IR}) \wedge \text{participates}(i, o_2))$
Process Ontology	None
Goal	$(\forall i\, \text{item}(i) \supset \text{GR\_IR\_Rule}(i))$

In this scenario, the data theory articulates a compliance rule of recorded goods receipts requiring an accompanying invoice receipt, while the goal enforces this rule for all items present in the dataset. This is a very lightweight reasoning task, and provides a good example of the architecture's flexibility for the complexity of reasoning tasks. Additionally, the handling of this scenario is flexible with respect to the implemented reasoner, and whether or not it implements the closed-world (CWA) or unique-names assumption (UNA). In a datalog [25] implementation, for example, CWA means that compliance failures for this goal will be detected via logical inconsistencies, while a reasoning implementation without CWA will not be inconsistent and will instead require additional considerations including identifying the existentially quantified "anonymous" individuals.

Note also that the goal depicted here does not exist in a vacuum - common sense may dictate that, in addition to requiring at least one accompanying invoice receipt, there should be no more than one goods receipt. While these rules could be merged into a singular rule, we instead adopt a more minimal approach in organizing the rules of compliance, such that uniqueness of these receipts could be an additional rule added as part of a goal if desired.

4.3. Ping-Pong/Multi-Hop Behaviour Detection

This scenario is taken from a log from Volvo IT Belgium, where the log depicts the IT incident management process [26]. The scenario here involves detecting "ping-pong" (also often called multi-hop) behaviour, where a process execution is passed amongst many resources. The data and representation are in tables 5 and 6.

Table 5. Data for identifying hand-offs

Activity	**Resource**	**Timestamp**	Case
Accepted - In Progress	Org Line C	16:28	A
Queued - Awaiting Assignment	Org Line B	16:35	A
Accepted - Wait-User	Org Line C	16:40	A

[1] Since these process ontology axioms are verbose, we reference their definition rather than explicitly define them here. They are included at `tinyurl.com/prover9examples`

Table 6. Hand-Off Example

Domain Knowledge	Hand-offs occur when subsequent activities are executed by different resources. Ping-pong behaviour occurs when a case has several hand-offs.
Reasoning Problem	Identifying all instances of ping-pong behaviour.
Mapped Sentence	$\big($subactivity_occ$(o_1, A) \wedge$ subactivity_occ(o_2, A) $\wedge$ subactivity_occ$(o_3, A) \wedge$ occurrence_of$(o_1,$ Acc.IPr$)\wedge$ occurrence_of$(o_2,$ Q.Aw$) \wedge$ occurrence_of$(o_3,$ Acc.WU$)\wedge$ participates$(\text{OrgC}, o_1) \wedge$ participates$(\text{OrgB}, o_2)\wedge$ participates$(\text{OrgC}, o_3) \wedge$ occurs_at$(o_1,$ `16:28`$)\wedge$ occurs_at$(o_2,$ `16:35`$) \wedge$ occurs_at$(o_3,$ `16:40`$) \wedge$ before$($ `16:28`, `16:35`$) \wedge$ before$($`16:35`, `16:40`$)\big)$
Data Theory	$\big(\forall(r_1, r_2, o_1, o_2, c)$ hand_off$(r_1, r_2, o_1, o_2, c) \equiv$ next_subOcc$(o_1, o_2, c) \wedge$ participates$(r_1, o_1)\wedge$ participates$(r_2, o_2) \wedge (r_1 \neq r_2)\big)$; $\big(\forall(c)$ *ping_pong*$(c) \equiv \exists(e_1, e_2, e_3, e_4, r_1, r_2)$ hand_off$(e_1, r_1, e_2, r_2, c)\wedge$ hand_off$(e_3, r_3, e_4, r_4, c) \wedge (e_1 \neq e_3) \wedge (e_2 \neq e_4)\big)$
Process Ontology	next_subOcc definition, transitivity of before, activity occurrence ordering definition [1]
Goal	$\big(\exists c$ ping_pong(c)$\big)$

In this scenario, the data theory is applied for a similar purpose in defining a pattern of behaviour in the process. However, in this case, the goal being existentially quantified amounts to identifying all the instances of the behaviour rather than ensuring it is true in all cases - it is the difference between a query and validating constraint. In the proof of this goal, the satisfying instances of the ping-pong definition correspond to each combination of hand-offs within cases, clearly identifying the reasoning behind their classification. Furthermore, modifications of this definition are straightforward to make while maintaining this transparency. Additionally, the process ontology is necessary to prove this goal since it relies on inferring the correct ordering of subactivites based only on their timestamps.

In this real scenario, the scope of "ping-pong" behaviour is further restricted to only consider non-system hand-offs. In this dataset, any time a work item is placed in a queue, logged event will show as the same resource, regardless of the organizational level that resource is in. For instance, if Alice in Line C needs to pause her current ticket, a "queued" event will be logged by Line B, a system resource - control of the work item being has not really changed. In the status quo of process mining, this will simply be handled by removing or manually overriding data that does not match this narrative, thus obfuscating key process knowledge being applied. With our architecture, the data theory provides the capability to formally capture this knowledge. New definitions for hand-off and ping-pong behaviour can be expressed, which subsume the previously defined ones, where terms like $\neg$(occurrence_of$(o_2, a) \wedge$ system_queue_activity(a)) can be added to specify this distinction.

These real and grounded scenarios provide evidence of our architecture's ability to handle real process challenges and exemplify the novel positioning of the process data theory, ontology, and mapped data. Furthermore, we have shown that the reasoning capabilities necessary to solve these problems are attainable using our architecture with an existing process ontology, and that this reasoning can be tuned according to the knowledge necessary to employ them - from the case of a singular axiom in the process ontology up to the combination of several axioms from distinct modules of the ontology with domain knowledge, they all provide clear answers to real process mining problems.

5. Conclusion: Operational Realization, Benchmarking, and Beyond

In this paper, we have specified a meaningful and novel challenge for existing process ontologies to become more data-driven and application-focused, understood through the new term: *operational realization*. We address this challenge with the introduction of an architecture that integrates formal process ontologies for practical use in data-driven environments, particularly process mining. The key novelty in this architecture includes the concept of a *data theory* that enriches the A-Box and T-box distinction in data-driven scenarios. The data theory enables the expression of knowledge often overlooked as part of ad-hoc data interpretation, and includes a novel mapping structure that leverages OBDA tooling while supporting more expressive mappings and reasoning capabilities beyond data access. However, several challenges remain to realize the full vision of this architecture in practice, particularly in scalability for large datasets and the packaging of the software tools necessary to support the architecture.

5.1. Implementation and Challenges

Key elements of our architecture have been implemented and made openly available,[1] demonstrating its application using real enterprise data across diverse domains, with widely utilized tools including Python and Datalog. The remaining challenges of scalability of reasoning are owed to the fact that reasoning with expressive ontologies necessarily introduces computational complexity [27]. In addition, more precise formalization of process data theory and reasoning templates as an integral and reified part of a software implementation would provide greater clarity and accessibility to end-users. Possible directions for these challenges are along a related path, as refining the kinds of permitted logical sentences in the data theory and for reasoning controls the complexity of reasoning tasks. Future work will explore the relationship between the sub-theories of an ontology which are necessary for specific reasoning tasks, and these observations can inform the structure of a library of reasoning and querying patterns.

Additionally, the output of automated reasoners is often suitable, albeit complex and/or verbose, for users with a background in mathematical logics. However, to be suitable for process mining practitioner's usage, these outputs should be integrated into a format more similar to emergent work in object-centric process querying libraries [28]. The current implementation will need to become both more automated in the deployment of automated reasoning as well as employ new and more expressive means of reasoning in order to solve these challenges. Preliminary development and testing results have shown promise in using the Z3 SMT solver [29] (and its python API) for this purpose.

[1] https://purl.org/ontologydrivenprocessanalysis/fois25

5.2. Process Ontology Benchmarking and Takeaways

A key role of this architecture will be as foundation for a process ontology benchmarking environment, enabling assessment of process ontologies in grounded and data-driven scenarios. While in this paper we have demonstrated key elements of the architecture through grounded problems, a process ontology benchmarking environment would include a curated library of challenge problems, with our architecture serving as a guide to apply a given process ontology. These challenge problems can be categorized based not only on their domain and data, but also in the kinds of reasoning tasks performed, similar to the categorization provided in this paper's validation. While both the aim of benchmarking ontologies and the concept of operational realization are not unique to the process domain, and could be extended to foundational ontologies more broadly, our focus is currently on processes. The prescriptive nature and the structured ways processes are modelled and analyzed make them especially well-suited to the archetypal patterns and reasoning tasks captured by our framework.

For the formal ontology community, this results in more grounded and practically applicable ontologies that have a basis in practical domain requirements. For process mining practitioners, this lays the foundation for further integration of process ontologies into their workflows, enhancing transparency, verifiability, and replicability.

References

[1] Van Der Aalst W, Adriansyah A, De Medeiros AKA, Arcieri F, Baier T, Blickle T, et al. Process mining manifesto. In: Business Process Management Workshops: BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I 9. Springer; 2012. p. 169-94.

[2] van der Aalst W. Process mining: a 360 degree overview. In: Process Mining Handbook. Springer; 2022. p. 3-34.

[3] Suriadi S, Andrews R, ter Hofstede A, Wynn MT. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. Information systems. 2017;64:132-50.

[4] Lopes IF, Ferreira DR. A survey of process mining competitions: the BPI challenges 2011–2018. In: Business Process Management Workshops: BPM 2019 International Workshops, Vienna, Austria, September 1–6, 2019, Revised Selected Papers 17. Springer; 2019. p. 263-74.

[5] Pedrinaci C, Domingue J. Towards an ontology for process monitoring and mining. In: CEUR Workshop Proceedings. vol. 251; 2007. p. 76-87.

[6] Norton B, Cabral L, Nitzsche J. Ontology-based translation of business process models. In: 2009 Fourth International Conference on Internet and Web Applications and Services. IEEE; 2009. p. 481-6.

[7] Borgo S, Ferrario R, Gangemi A, Guarino N, Masolo C, Porello D, et al. DOLCE: A descriptive ontology for linguistic and cognitive engineering. Applied ontology. 2022;17(1):45-69.

[8] Guizzardi G, Botti Benevides A, Fonseca CM, Porello D, Almeida JPA, Prince Sales T. UFO: Unified foundational ontology. Applied ontology. 2022;17(1):167-210.

[9] Xiao G, Calvanese D, Kontchakov R, Lembo D, Poggi A, Rosati R, et al. Ontology-based data access: A survey. International Joint Conferences on Artificial Intelligence; 2018. .

[10] Gunther CW, Verbeek H. Xes-standard definition. 2014.

[11] Ghahfarokhi AF, Park G, Berti A, van der Aalst WM. OCEL: A standard for object-centric event logs. In: European Conference on Advances in Databases and Information Systems. Springer; 2021. p. 169-75.

[12] Fahland D. Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process mining handbook. Springer; 2022. p. 274-319.

[13] Swevels A, Fahland D, Montali M. Implementing object-centric event data models in event knowledge graphs. In: International Conference on Process Mining. Springer; 2023. p. 431-43.

[14] Di Ciccio C, Montali M, et al. Declarative Process Specifications: Reasoning, Discovery, Monitoring. Process mining handbook. 2022;448:108-52.
[15] Bergami G, Maggi FM, Marrella A, Montali M. Aligning data-aware declarative process models and event logs. In: Business Process Management: 19th International Conference, BPM 2021, Rome, Italy, September 06–10, 2021, Proceedings 19. Springer; 2021. p. 235-51.
[16] Burattin A, Maggi FM, Sperduti A. Conformance checking based on multi-perspective declarative process models. Expert systems with applications. 2016;65:194-211.
[17] Calvanese D, Kalayci TE, Montali M, Tinella S. Ontology-based data access for extracting event logs from legacy data: the onprom tool and methodology. In: Business Information Systems: 20th International Conference, BIS 2017, Poznan, Poland, June 28–30, 2017, Proceedings 20. Springer; 2017. p. 220-36.
[18] Moher R, Gruninger M. Mining for Meaning: Ontology-Aware Process Mining Methods Through Knowledge Patterns. In: International Conference on Research Challenges in Information Science. Springer; 2025. p. 109-19.
[19] Grüninger M. Ontology of the process specification language. In: Handbook on ontologies. Springer; 2004. p. 575-92.
[20] Botoeva E, Calvanese D, Cogrel B, Rezk M, Xiao G. OBDA beyond relational DBs: A study for MongoDB. In: CEUR Workshop Proceedings. vol. 1577. CEUR-WS. org; 2016. .
[21] Thiede M, Fuerstenau D, Bezerra Barquet AP. How is process mining technology used by organizations? A systematic literature review of empirical studies. Business Process Management Journal. 2018;24(4):900-22.
[22] Heyvaert P, De Meester B, Dimou A, Verborgh R. Declarative rules for linked data generation at your fingertips! In: The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15. Springer; 2018. p. 213-7.
[23] van Dongen B. BPI Challenge 2012. Eindhoven University of Technology; 2012.
[24] Augusto A, Leno V, Reissner D. BPI Challenge 2019 Report: a Purchase-to-Pay Process Analysis. University of Melbourne. 2019.
[25] Maier D, Tekle KT, Kifer M, Warren DS. Datalog: concepts, history, and outlook. In: Declarative Logic Programming: Theory, Systems, and Applications; 2018. p. 3-100.
[26] Steeman W. BPI Challenge 2013, incidents. Ghent University; 2013.
[27] Madelaine F, Martin B. On the complexity of the model checking problem. arXiv preprint arXiv:12106893. 2012.
[28] Küsters A, van der Aalst WM. OCPQ: Object-Centric Process Querying and Constraints. In: International Conference on Research Challenges in Information Science. Springer; 2025. p. 383-400.
[29] De Moura L, Bjørner N. Z3: An efficient SMT solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer; 2008. p. 337-40.